TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ILARI KAMPMAN

ALGORITHMS FOR CLUSTERING HIGH-DIMENSIONAL DATA

Master of Science thesis

Examiner: Prof. Tapio Elomaa
Examiner and topic approved by the Faculty Council of the Faculty of Natural Sciences
on 27th September 2017

ABSTRACT

ILARI KAMPMAN: Algorithms for Clustering High-Dimensional Data
Tampere University of Technology
Master of Science thesis, 59 pages, 17 Appendix pages
December 2017
Master of Science (Tech) Degree Programme in Science and Engineering
Major: Mathematics
Examiner: Prof. Tapio Elomaa
Keywords: Machine Learning, Clustering, High-dimensional Data, Principal Component Analysis

In the context of artificial intelligence, clustering refers to a machine learning method in which a data set is grouped into subsets based on similarities in the features of data samples. Since the concept of clustering is broad and complex, clustering algorithms are often designed for solving specific problems. Especially, for overcoming the challenges proposed by high-dimensional data, specialized techniques are required. The existing algorithms for clustering high-dimensional data are often too dependent on prior information about the data, which is why they are not necessarily suitable for autonomous applications of artificial intelligence.

This thesis introduces two new algorithms for clustering high-dimensional shapes: CHUNX and CRUSHES. The clusters of both algorithms are based on a hierarchical tree structure which is formed using matrix diagonalization done in Principal Component Analysis (PCA). Due to its expressiveness, the constructed tree structure can be used efficiently for detecting noise clusters and outliers from the data. In addition, the clustering parameter setup in CHUNX and CRUSHES is more flexible than in other PCA-based clustering algorithms.

In the empirical part of this thesis, CHUNX and CRUSHES are compared with k-means, DBSCAN and ORCLUS clustering algorithms. The algorithms are evaluated by their usability in situations where there is no prior information about the structure of data to be clustered. The data used in the empirical algorithm evaluation consists of energy curve shapes from the Finnish electrical grid. According to the algorithm evaluation, CHUNX and CRUSHES produce more detailed results and are more suitable for situations, where there is no prior information about the data, than the other algorithms of the evaluation.

TIIVISTELMÄ

ILARI KAMPMAN: Algoritmeja moniulotteisen datan klusterointiin
Tampereen teknillinen yliopisto
Diplomityö, 59 sivua, 17 liitesivua
joulukuu 2017
Teknis-luonnontieteellinen DI-tutkinto-ohjelma
Pääaine: Matematiikka
Tarkastajat: Prof. Tapio Elomaa
Avainsanat: Koneoppiminen, Klusterointi, Moniulotteinen data, Pääkomponenttianalyysi

Tekoälytutkimuksessa klusteroinnilla tarkoitetaan koneoppimismenetelmää, jolla datajoukko voidaan ryhmitellä osajoukkoihin datahavaintojen ominaisuuksien samankaltaisuuden perusteella. Klusterointi on laaja ja monisyinen käsite, minkä vuoksi klusterointialgoritmit on suunniteltu ratkaisemaan tarkasti rajattuja ongelmia. Erityisesti moniulotteisen datan klusterointiin liittyy haasteita, joiden ratkaisemiseksi tarvitaan erikoistuneita menetelmiä. Olemassa olevat, moniulotteisen datan klusterointiin tarkoitetut algoritmit nojaavat usein liikaa datasta saatuihin ennakkotietoihin, minkä vuoksi ne eivät välttämättä sovellu osaksi autonomisia tekoälysovelluksia.

Tässä työssä esitellään kaksi uutta algoritmia moniulotteisten muotojen klusterointiin: CHUNX ja CRUSHES. Algoritmien tuottamat klusterit perustuvat pääkomponenttianalyysissa tehtävän matriisin diagonalisoinnin avulla muodostettavaan hierarkiseen puurakenteeseen – ilmaisuvoimaista puurakennetta voidaan hyödyntää erityisesti poikkeavien datahavaintojen ja klustereiden tunnistamisessa. Lisäksi CHUNX- ja CRUSHES-algoritmien klusterointiparametrien asettelu on joustavampaa kuin muissa pääkomponenttianalyysiin perustuvissa klusterointialgoritmeissa.

Työn kokeellisessa osiossa CHUNX- ja CRUSHES-algoritmeja verrataan k-means-, DBSCAN- ja ORCLUS-algoritmeihin. Vertailu painottuu algoritmien käytettävyyteen tapauksessa, jossa datan rakenteesta ei ole ennakkotietoja saatavilla. Kokeellisena datana vertailussa käytetään suomalaisen sähköverkon sähkönkulutuskäyrien muotoja. Vertailun perusteella CHUNX- ja CRUSHES-algoritmit tuottavat yksityiskohtaisempia tuloksia ja soveltuvat vertailualgoritmeja paremmin tapauksiin, joissa datan rakennetta ei etukäteen tunneta.

PREFACE

This thesis was carried out as a research project for my current employer, *Wapice Ltd*, during the year 2017. The initial objective of the project was to investigate, if the use of machine learning techniques could improve the accuracy of energy consumption forecasts in *EcoReaction*, which is a solution for Consumption Information Management. Due to the challenging nature of high-dimensional energy consumption data, the research focused on ways to cluster the given EcoReaction data.

First I would like to thank Jukka Hämäläinen and Juha Heikkinen from the EcoReaction development team for the great opportunities to explore real-life data and to develop new data analysis features for EcoReaction. I am equally grateful for my colleague Robert Yates for his valuable advice on performing numerical analysis on large sets of data. I would also like thank Professor Tapio Elomaa for his constructive feedback, flexibility and encouragement during the writing process of my thesis.

In addition, I would like to thank my friend Valtteri Peltonen for the discussions we have had about the mathematics of my thesis, since they have been a great help in the development of my algorithms. Finally, words cannot express the gratitude to my dear wife Karoliina and my dear daughter Saaga for the endless support and happiness in my life – you have been the driving force behind my studies.

Tampere, on 21st November 2017

Ilari Kampman

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

$n \in \mathbb{N}$	A natural number
$A \in \mathbb{R}$	A set of real numbers
$\boldsymbol{x} \in \mathbb{R}^n$	A vector in n-dimensional real space
$\boldsymbol{X} \in \mathbb{R}^{n \times d}$	A real matrix with n rows and d columns
$\boldsymbol{x}^T$	A transpose of vector $\boldsymbol{x}$
$\boldsymbol{X}^T$	A transpose of matrix $\boldsymbol{X}$
$\sum_{i=1}^{n} a_i$	The sum of elements a_i where $i = 1, \ldots, n$
$\binom{a}{b}$	The binomial coefficient which is equal to the number of b-combinations from the set of a elements
$\langle \cdot, \cdot \rangle$	The inner product
$\operatorname{corr}(\cdot, \cdot)$	Correlation of two vectors
$\operatorname{corr}(\cdot)$	Correlation of a matrix
$\operatorname{col}(\cdot)$	The set of column vectors of a matrix
$\mathbf{1}_n$	A vector of n elements all of which have the value 1
$\operatorname{cov}(\cdot, \cdot)$	Sample covariance of two vectors
$\operatorname{row}(\cdot)$	The set of row vectors of a matrix
$\operatorname{var}(\cdot)$	Sample variance of a vector
$\operatorname{cov}(\cdot)$	Sample covariance of a matrix
$\boldsymbol{X}_c$	Centered data matrix
$\boldsymbol{\Sigma}_{\boldsymbol{A}}$	Sample covariance matrix of matrix $\boldsymbol{A}$
λ	An eigenvalue of a matrix
$\det(\cdot)$	Determinant of a matrix
$\boldsymbol{X}^{-1}$	Inverse of matrix $\boldsymbol{X}$
$\boldsymbol{I}$	Indentity matrix
$\hat{\boldsymbol{X}}_c$	Centered data matrix with length normalized row vectors
$\boldsymbol{X}^2$	A matrix whose elements are squared

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CHUNX	Clustering Hierarchical correlations of cUrve shapes uNtil clusters satisfy maXimum size criterion
CRUSHES	CoRrelation clUStering based on Hierarchical mutual correlation Estimation of curve Shapes
PCA	Principal Component Analysis
DBSCAN	Density-Based Spatial Clustering of Applications with Noise algorithm
STING	STatistical INformation Grid
ANN	Artificial Neural Network
SOM	Self-Organized feature Map
CNN	Convolutional Neural Network
ORCLUS	arbitrarily ORiented projected CLUSter generation
UTC	Coordinated Universal Time

1. INTRODUCTION

During the last decade the use of *Machine Learning* techniques in the analysis of large-scale data has rocketed not just in every field of engineering but in almost every field of science as well [47]. There seems to be no saturation for the exponential growth of the numerous large data sets belonging to broad spectrum of actors from devices of regular people to the broad data acquisition systems of global enterprises, universities, and governments of states world wide [17, 39]. The unlimited resources of data have encouraged different data collectors to make predictions continuously about the history data using more and more complex machine learning methods. An automated process capable of reasoning, which resembles human thinking, is referred as an *Artificial Intelligence* (AI) system. Over the last five years the fast development of computational technologies has opened possibilities for applications of AI which use computationally expensive machine learning algorithms.

Ever since the 1950s, long before the modern computer technology, machine learning has been an important topic in the research of AI [60]. Machine learning, in the field of AI, has been described as the computer's capabilities of adaptation under changing circumstances and distinguishing characteristic patterns or similarities within data. Since AI tries to imitate the processes of human mind, learners and other units of rational acting are referred as *agents* in the field of AI. In machine learning, the tasks given to an actor are often classified under three main-categories: *supervised learning*, *reinforcement learning*, and *unsupervised learning*. The category of a specific learning task is determined based on the type of feedback the agent receives. In supervised learning, the inference of the output value on certain input is done based on previous observations of input-output pairs given to an agent as predefined feedback. Reinforcement learning is based on rewards or punishments given to an agent as an instant feedback after acting over the learning process. Unlike in supervised learning, where the agent has generated a static predictive function, the reinforcement learning is a gradual, iterative process. The category of unsupervised learning consists of tasks in which the agent gets no feedback but detects distinguishable patterns from the input. Since the categories of machine learning are quite strictly defined, there exists methods which fall into the category of *semi-supervised learning*, in which the agent has to deal with incomplete or false

explicit feedback. In practical applications, it is often hard or even impossible to define, in which output category a certain input belongs to or how many categories there are in total. Sometimes the goal is to let the agent detect whether some features are relevant or not, which makes the study of unsupervised learning methods important. [51, p. 2, 694–695]

Clustering is the most common unsupervised learning technique [51, p. 695]. It has been studied broadly not just in the field of machine learning but in the field of data mining [11] and in the applications of different sciences from Earth sciences to Medicine as well [8]. Clustering aims at partitioning the given data into characteristic subsets in which the members are mutually as similar as possible. The use of clustering has traditionally focused on Euclidean space applications of different fields of study where the distances between data points can be visualized in spaces which have two or three dimensions [37]. In the recent years, the subject of high-dimensional clustering has been studied actively [37]. As the number of dimensions in data rises, traditional similarity measures and clustering algorithms utilizing them usually become meaningless and the visualization of distances between data points becomes harder [37]. The set of problems the clustering of high-dimensional data faces is often referred as *The Curse of Dimensionality* [10, 37].

This thesis introduces two new *correlation clustering* algorithms called *Clustering Hierarchical correlations of cUrve shapes uNtil clusters satisfy maXimum size criterion* (CHUNX) and *CoRrelation clUStering based on Hierarchical mutual correlation Estimation of curve Shapes* (CRUSHES) which are based on *Principal Component Analysis* (PCA). The algorithms are suitable especially for high-dimensional data which has multiple, variable-size subsets with distinguishable characteristic behavior. The main objective for the development of CHUNX and CRUSHES algorithms has been to detect subgroups of characteristic behavior from a large data set of time-dependent time series. Although there are distinguishable subgroups in the data, a large proportion of samples have a similar overall shape. It can be hard to determine subgroups from high-dimensional data in which a single feature dominates. CHUNX and CRUSHES try to solve this problem by representing each data point using a finite number of each data point's most significant components computed in PCA. Both algorithms produce a hierarchical cluster structure according to variations of components in data point representations.

Chapter 2 provides an overview on the earlier work related to the subject of clustering algorithms. The chapter explains briefly a categorization of clustering algorithms with the help of example algorithms from certain categories. The focus of Chapter 2 is on the challenges high-dimensional data proposes to clustering and on the features

of different algorithms trying to overcome the challenges.

Chapter 3 describes CHUNX and CRUSHES algorithms for clustering high-dimensional data in detail. The chapter includes mathematical background, descriptions, and pseudocode representations of both CHUNX and CRUSHES.

Chapter 4 evaluates the results of CHUNX and CRUSHES algorithms for a large set of real-life electricity grid energy consumption data. The results of CHUNX and CRUSHES are evaluated by comparing the user experiences and the results of different existing clustering algorithms. CHUNX and CRUSHES are compared not only by the results they produce but by their sensitivity to different algorithm parameter settings.

The final chapter concludes the main achievements and the results of this thesis. The chapter's discussion focuses on the benefits and drawbacks of CHUNX and CRUSHES algorithms but provides also ideas for algorithm improvements and future work related to the subject of this thesis.

2. RELATED WORK

Clustering, the most common unsupervised learning technique, tries to partition a set of data into smaller subsets of similar data based on the internal features of data [26, p. 383]. Even though the general level definition of clustering is not strict, there are few points that traditionally have defined clustering [46]. According to the traditional definition, the elements in a cluster are as similar as possible, the elements in different clusters are as dissimilar as possible, and the measure of similarity or dissimilarity between elements must be intuitive [26, 64]. On the other hand, clustering can be used as method for detecting noise or outliers – observations in data which are clearly dissimilar from other observations. The reason that there is no consensus about the definition of clustering lies in the broad spectrum of challenges and settings different applications present [26].

Even though clustering algorithms are often specialized in a focused problem setting due to the wide variety of clustering applications, there are some common features that are often required. In order to solve a real-life problem by clustering, the algorithm used must be usable without intensive study of an appropriate hyper-parameter setting. Complicated hyperparameter requirements not just make the algorithm hard to use but often make the clustering result hard to control. Another algorithm usage related requirement for a clustering algorithm is to produce results that are comprehensible and practical. [26]

Since clustering algorithms are designed to be as general as possible, they must determine the types and shapes of data it can be used for. The data used in clustering is often numerical but some applications require categorical, binary, ordinal or a combination of these formats to be clustered. Clustering is often performed for data that cannot be easily classified into categories due to the complexity of the relationships between similar observations or due to the large scale of observations. Therefore, the algorithms must be able to detect clusters with arbitrary shapes in variable size data sets. It is often required from an algorithm to able to distinguish noise from actual clusters too. [26]

One of the most difficult requirements a clustering algorithm has to satisfy is the

ability to cluster high-dimensional data. There are many clustering algorithms that cluster two- or three-dimensional data producing quality results but fail when the number of dimensions gets high. Many clustering algorithms aim at satisfying the most of these requirements, but due to different problem settings, the algorithms have to focus on few of these requirements in order to solve specialized and application oriented problems. [26]

2.1 Measuring Dissimilarity

In order to partition a set of data D of n d-dimensional data points, where $n \in \mathbb{N}$ and $d \in \mathbb{N}$, a clustering algorithm must either get dissimilarity information about D as a precomputed input parameter or the dissimilarity structure must be inferred within the algorithm. In a situation where the dissimilarity information of D is precomputed, it is common to pass the information to an algorithm as a *dissimilarity matrix*

$$\boldsymbol{D} = \begin{bmatrix} 0 & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & 0 & \cdots & \vdots \\ \vdots & \vdots & \ddots & d_{n-1,n} \\ d_{n,1} & d_{n,2} & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{2.1}$$

where $d_{i,j}$ represents the the dissimilarity of the i^{th} and j^{th} data points, $d_{i,j} = d_{j,i}$, $d_{i,i} = 0$, and $i, j \in \{1, 2, \ldots, n\}$. If the similarity of the i^{th} and j^{th} data points is high, the value of $d_{i,j}$ is close to 0. In a situation, where the clustering algorithm does not operate on precomputed dissimilarity information, D must be represented as a *data matrix*

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix}^T = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \in \mathbb{R}^{n \times d}, \tag{2.2}$$

where $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, n$, and operator T denotes the *transpose* of a matrix. Each row in $\boldsymbol{X}$ represents a d-dimensional data point. The rows of $\boldsymbol{X}$ are sometimes referred as *samples* and the columns as *attributes* [26]. A dissimilarity matrix can be computed from a data matrix using different measures of dissimilarity.

Over the last century, there has been at least 76 different measures of binary dissimilarity or similarity and distance used in scientific literature [16]. The most common measures of distance used in clustering are *Euclidean distance* [16] and *Manhattan distance* [35]. Euclidean distance refers to the shortest distance between two points

in d-dimensional space,

$$d_{\text{Euclidean}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2}, \tag{2.3}$$

which is also the L_2-norm of vector $\boldsymbol{x}_i - \boldsymbol{x}_j$. When the number of dimensions is at most three, Euclidean distance is the most intuitive measure of distance to visualize and understand. Manhattan distance or *taxicab distance* is the shortest distance between data points along the coordinate axes in d-dimensional spaces,

$$d_{\text{Manhattan}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1 = \sum_{k=1}^{d}|x_{i,k} - x_{j,k}|,$$

which is also the L_1-norm of vector $\boldsymbol{x}_i - \boldsymbol{x}_j$. Another commonly used measure of distance is *Chebyshev distance*,

$$d_{\text{Chebyshev}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_\infty = \max_k |x_{i,k} - x_{j,k}|,$$

which is the maximum distance between vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in any dimension – the L_∞-norm of vector $\boldsymbol{x}_i - \boldsymbol{x}_j$ [15, p. 301].

Rather than measuring distance of vectors by computing the difference of a chosen norm of a vector, *cosine distance* or *correlation* measures the angle between vectors,

$$d_{\text{cosine}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \text{corr}\,(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\|\boldsymbol{x}_i\|_2 \|\boldsymbol{x}_j\|_2} = \frac{\sum_{k=1}^{d} x_{i,k} x_{j,k}}{\|\boldsymbol{x}_i\|_2 \|\boldsymbol{x}_j\|_2}, \tag{2.4}$$

where operator $\langle \cdot, \cdot \rangle$ denotes the *inner product* and $\text{corr}\,(\cdot, \cdot)$ denotes the *correlation* of vectors [15, p. 302]. Correlation between vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is denoted by $\rho_{i,j}$. It should be noted that correlation of 1 represents the angle of 0 radians and correlation of -1 the angle of π radians. In order to use correlation distance data as in Equation (2.1), the distances of vectors must be represented as the angles between vectors. This can be done by taking the arcus cosine of the correlation distance data,

$$\boldsymbol{D}_{\text{angles}} = \arccos\big(\text{corr}\,(\boldsymbol{X})\big) = \arccos\left(\begin{bmatrix} 1 & \rho_{1,2} & \cdots & \rho_{1,n} \\ \rho_{2,1} & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \rho_{n-1,n} \\ \rho_{n,1} & \rho_{n,2} & \cdots & 1 \end{bmatrix}\right), \tag{2.5}$$

where $\text{corr}\,(\boldsymbol{X})$ denotes the *correlation matrix* of $\boldsymbol{X}$ whose diagonal elements are 1

and other elements $\rho_{i,j} \in [-1, 1]$, where $i, j \in \{1, \ldots, n\}$, represent the correlations between row vectors of $\boldsymbol{X}$.

Euclidean, Manhattan, Chebyshev, and cosine distances satisfy the properties of a *metric*. A metric is a nonnegative function $d : X \times X \to [0, \infty)$ for set X, where for all $x, y, z \in X$

$$d(x, x) = 0, \tag{2.6}$$

function d is *symmetric*,

$$d(x, y) = d(y, x),$$

and satisfies the *triangle inequality*,

$$d(x, z) \leq d(x, y) + d(y, z).$$

By Equation (2.6), if $d(x, y) = 0$, then $x = y$ [1].

2.2 Clustering Algorithms

The number of existing clustering algorithms is large and the purpose of clustering varies a lot in applications. As some of the clustering algorithms share similar features but produce completely different results, it can be difficult to categorize a clustering algorithm unambiguously. It is actually quite rare that a clustering algorithm would strictly belong to a certain category. There are different ways to categorize the main features of clustering algorithms [22, 26, 64]. In this thesis the cluster categorization presented in the textbook of Han and Kamber [26] is used. The main categories of clustering are: *partitioning methods*, *hierarchical methods*, *density-based methods*, *grid-based methods*, *model-based methods*, *constraint-based methods*, and *clustering high-dimensional data*. The characteristic features of each main category will be described this section with example algorithms from chosen categories.

Partitioning clustering algorithms partition a data set D of n elements into k strict partitions, where $n \geq k$. In general, each of the k clusters must contain at least one element and each element in D must belong to a single cluster. The main goal of a partitioning algorithm is to optimize the function used for measuring total similarity within clusters or dissimilarity between clusters. The partitioning algorithms usually start by initializing a partitioning which is improved through iterations in which the initial partitions are modified until a satisfactory partitioning is reached. Since the problems of finding the optimal partitioning of D into k partitions in 2-dimensional Euclidean space and into two partitions in d dimensions are NP-hard [24], there are

heuristic methods for partitioning a data set efficiently. [26]

The most famous and the most common partitioning and clustering algorithm, *k-means* [40], partitions a data set D of n into a chosen number of k partitions. In the beginning of k-means algorithm, k data points from D are selected randomly as the initial centers of clusters. All the remaining data points in D are assigned to the cluster their distance is the smallest to. After the initial partitioning, the center of each cluster, *centroid*, is computed. This process continues to assign points and compute new centroids until a criterion function converges. The iterations of k-means is illustrated in Figure 2.1 Most typical stopping criterion used in k-means is

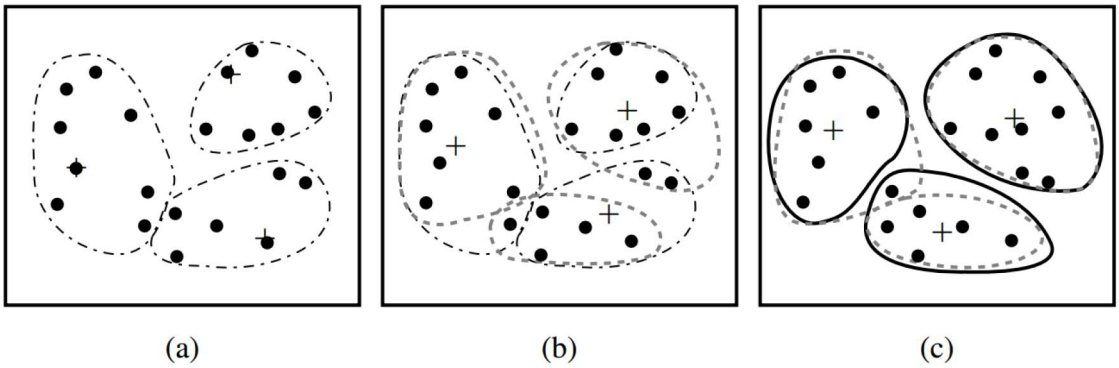

Figure 2.1 *The iterations of k-means algorithm: (a) The initial cluster partitioning, where centroids are marked with "+" (b) The partitioning after the centroid computation marked with darker dashed line (c) The assignment of points the clusters nearest to them marked with solid line [26, p. 403].*

the *square-error criterion* which refers to the sum of squared errors from the centroid $\boldsymbol{m}_i$ of a cluster C_i each element $\boldsymbol{x}$ belongs to,

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} |\boldsymbol{x} - \boldsymbol{m}_i|^2. \tag{2.7}$$

It should be noted that k-means produces clusters with spherical shape. In the implementations of k-means variants, the distance information of D is computed within the implementation from data matrix given in Equation (2.2) [54]. A common implementation of k-means called *Lloyd's algorithm* has a running time in $\Theta(dkni)$, where i refers to the number of iterations [42, p. 364].

Even though the partitioning algorithms are computationally efficient and widely used, they have drawbacks especially in the situations where the clusters are not clearly separate from each other or there are hierarchical structures within the clusters. The issues lie in the nondeterminism of partitioning algorithms: Since the

results of some of the partitioning algorithms varies a bit with every run, the clustering results can become hard to interpret [42, p. 377]. The hierarchical methods of clustering produce a hierarchy of elements within input data D often in a deterministic manner. The hierarchical methods are classified into *divisive* and *agglomerative* methods. The divisive, or top-down, methods start with the complete data D and proceed by dividing D recursively into subsets until there are only single-element clusters or predefined terminal conditions are satisfied. The agglomerative, or bottom-up, methods operate on D to the opposite direction: They start with single-element clusters and proceed by merging the clusters into larger clusters based on their similarity. Both the divisive and agglomerative methods often produce a tree-structure representing the individual relationships within the data. Even so, it is possible that the algorithms make a bad choice of dividing a cluster or merging clusters. Fortunately, the study of hierarchical clustering methods [66] has found ways to overcome the problems of straightforward merging of clusters. [26, p. 408–413]

Especially in the lower data dimensions, it is common that the input data is clearly clustered but the cluster shapes are arbitrary or some clusters surround other clusters. Even though there would be a fixed number of clusters in such data, common partitioning algorithms would not be able to detect the clusters since they operate on fixed cluster shapes [40]. The study of density-based clustering methods has developed methods for detecting clusters with arbitrary shape by calculating densities based on fixed density criteria [21], by creating an order of data points for density structure [9], and by modelling densities using a set of distribution functions [28].

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm has approached the density detection by defining a minimum number of points in one cluster $MitPts$ and the maximum radius of neighborhood for each data point ε [21]. DBSCAN iterates through the whole data set marking each visited data point. It starts by selecting a data point p and computes which data points are within the radius of ϵ from it. If the number of data points within the ε-neighborhood is less than $MinPts$, p is considered as noise [21, p. 228]. Otherwise, a point is considered as a *core point* and a cluster is formed. The formed cluster is expanded by computing the ε-neighborhood for all the data points in the ε-neighborhood of p. A data point q within the ε-neighborhood of p will be considered as a core point and the points within its ε-neighborhood belonging to the same cluster if the number of points in its ε-neighborhood is greater or equal to $MinPts$. Otherwise q will belong to the same cluster with p but it will be considered as a *border point* and the points within its ε-neighborhood won't necessarily belong to the same cluster with q. DBSCAN continues forming clusters and expanding them until every data point is marked as

visited. The cluster formation of DBSCAN is illustrated in Figure 2.2. DBSCAN

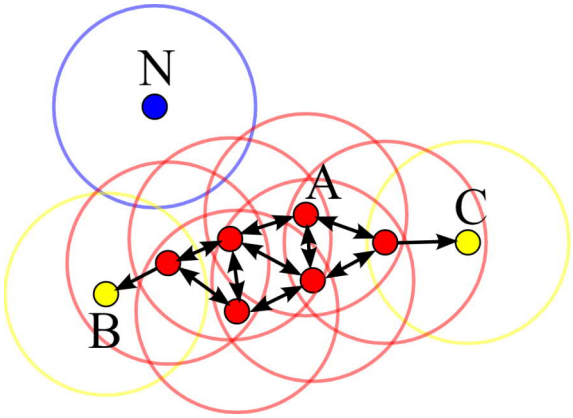

Figure 2.2 *The illustration of DBSCAN algorithm's cluster formation with $MinPts = 3$, where the red points marked with "A" are the core points, the yellow points marked with "B" and "C" are the border points, and the blue point marked with "N" is noise [63].*

has an average run time complexity in $O(n \log n)$ and the worst-case complexity in $O(n^2)$ [21].

In the grid-based methods of clustering, the object space of data is somehow partitioned into a grid of cells. In other words, the continuous numeric data is mapped into a discrete form. The grid-based methods often offer a possibility to investigate the internal hierarchical structures of data formed by the grid [26, p. 424–428]. Depending on the grid-based method, the shape of a grid cell and the procedures performed for a group of cells vary a lot. One of the most intuitive ways of forming a grid is used in *Statistical Information Grid* (STING) algorithm where a spatial area is divided into cells with rectangular shape. Each cell is then divided recursively into rectangular cells forming different layers of resolution for data. For each cell a set of statistical parameters are computed and stored. The statistical information is useful for processing queries to the clustered data [62]. There also exists methods which summarize a generated grid structure by projecting the original data using a transformation function such as *Wavelet transformation* in *WaveCluster* algorithm [55].

The model-based methods of clustering rely on some assumption about the underlying model of the input data such as a certain probabilistic function [18, 58] or even an *artificial neural network* (ANN) [14, 34]. In the *Expectation Maximization* approach of model-based clustering, each of the k clusters in data is expected to behave mathematically as a parametric probability distribution. Therefore, the complete data forms a finite *mixture density model* of k different probability distributions [26, p. 429]. The *Conceptual Clustering* approach implements the classification of data into the clustering process: Instead of considering the objects in a single cluster sim-

ilar, the conceptual clustering algorithms try to explain the characteristic features of each group [26, p. 431]. A common conceptual clustering method *COBWEB* algorithm [23] detects conceptual clusters by generating a *classification tree* from hierarchical clustering model. Another model-based approach, *Neural Network* approach, focused for several years almost only on the study of clustering with *Self-Organized feature Map* (SOM) ANNs due to computational limitations. In the recent years, as the computational resources have become more and more accessible, research has been made about the use of *Convolutional Neural Networks* (CNN) in model-based clustering as well [65].

Despite the major differences between the types of clustering, most of the algorithms discussed above share a common feature in their design: Apart from the individual parameter setups, the algorithms operate on data without application-specific constraints or additional user interaction. There are different ways to set constraints to resulting clusters by preprocessing the input data, filtering the clustering results itself or selecting a suitable range for a specific parameter of a clustering algorithm. The methods which take the application-specific constraints into account during the process of clustering are called constraint-based clustering methods – the constraining operations which are performed outside the clustering process are not considered as constraint-based methods. An example of constraint-based clustering called *Clustering with Obstacle Objects* is a method in which a distance function of a clustering algorithm is restricted to operate only on a predefined set of distances between data points. In other words, the data space can contain *obstacles* which have to be taken into account while computing distances between data points. These algorithms typically use a *visibility graph* for expressing the shortest valid distances between data points – the data points in the visibility graph may not be visible to each other at all. [26, p. 444–448]

2.3 Clustering in High Dimensions

Even though there is a broad spectrum of clustering methods from different types of approaches, a significant proportion of them have been designed to function only when the number of dimensions is low. As the number of dimensions in a clustering task is in hundreds or thousands, unintuitive problems start to arise and clustering often becomes cumbersome [37]. In high-dimensional data, it is usual that only a small number of dimensions is relevant in terms of clustering – the remaining dimensions tend to produce a lot of noise and mask the relevant features. High-dimensional data is often sparsely distributed and the most of the data vectors tend to be mutually orthogonal. The curse of dimensionality is a well-known set

of problems related to high-dimensional data in general [10]. In the concept of clustering, the curse of dimensionality can be highlighted with 4 basic problems [37, p. 43–46] which will be described in the following paragraphs.

Problem 1 The more there are dimensions, the more there are possibilities for values. Since clustering in general assumes that a data set can be generated by a finite number of functions, the functions mapping the data become more complex as the number of dimensions, function parameters, increases. Visualization of high-dimensional data is another topic related to this particular problem.

Problem 2 As the number of dimensions in data increases, the concepts of distance and proximity of two distinct data points become almost useless. It has been proven by Beyer et al. [12] that

$$\lim_{d\to\infty} \frac{D_{\max} - D_{\min}}{D_{\min}} = 0,$$

where d is the number of dimensions in the data set, $D_{\max}$ is the maximum distance, and $D_{\min}$ minimum distance between two data points in the data set. Therefore, using distance as a measure of similarity in high-dimensional spaces, it can be hard to say whether two data points are similar to each other or not.

Problem 3 The applications of clustering low-dimensional data aim at grouping data points whose attributes are the most similar to each other. As the number of dimensions increases, the data points can be related to each other by only a number of relevant attributes. The other remaining irrelevant attributes are considered as noise which often masks the relevant features.

Problem 4 Even though many high-dimensional clustering algorithms aim to extract the most significant attributes for clustering the data, it is possible that some attributes that are considered as noise are actual characteristic features of data. Therefore, attention must be paid when reducing noise in high-dimensional data.

The study of high-dimensional clustering has been very application-oriented and it has produced many highly effective and important algorithms for applications of genealogy, biology, and text analysis [7, 13, 19]. Even though the study has been highly active in the recent years, it has lacked a consistent approach to overcome the problems caused by the curse of dimensionality [37]. In the survey on clustering high-dimensional data, Kriegel et al. [37] present a comprehensive categorization of different high-dimensional clustering methods and give suggestions for problem-oriented study of high-dimensional clustering and standardized vocabulary to be used in related literature.

2.3.1 Clustering in Axis-Parallel Subspaces

According to the survey of Kriegel et al. [37, p. 9–20], a large number of existing high-dimensional clustering algorithms focus on *clustering in axis-parallel subspaces.* In this approach, the infinite search space of all possible subspaces is limited to subspaces which are parallel to dimesional axes. Even though the number of possible subspaces in this approach is finite, the number is still quite large in higher dimensions: For d-dimensional the number of all k-dimensional subsets is

$$\sum_{k=1}^{d} \binom{d}{k} = 2^d - 1,$$

where $1 \leq k \leq d$ and $d, k \in \mathbb{N}$. On the other hand, the finite search space can be very useful in applications where Problem 3 is on display.

The algorithms that cluster in axis-parallel subspaces can be categorized based on the assumptions they make about the underlying problem. The *Projected Clustering Algorithms* assume that every data point can be assigned uniquely to a single cluster or considered as noise. These algorithms produce a projection to clusters from the original data. Some projected clustering algorithms may assume that there is a fixed number of clusters to which the data can be optimally assigned. These algorithms are referred as *"Soft" Projected Clustering Algorithms*, since they are not producing hard unique assignments but optimized and approximate projections avoiding Problem 4. In another category called *Subspace Clustering Algorithms*, the algorithms compute all the possible subspaces where the clusters can be identified. The algorithms that combine the methods from the other categories of axis-parallel clustering are referred as *Hybrid Clustering Algorithms.* The algorithms within these categories can be again be categorized either as agglomerative or as divisive. [37, p. 10–12]

2.3.2 Clustering Based on Patterns in the Data Matrix

Clustering in axis-parallel subspaces sometimes collides with Problem 2 by determining the similarity of data points using a measure of distance or density. In the next category of high-dimensional clustering, called *Pattern-Based Clustering Algorithms*, the approaches look for characteristic behavior of data points concerning certain attributes of data. These algorithms also operate in axis-parallel subspaces. The algorithms which cluster data based on patterns in the data matrix are often referred as *biclustering algorithms.* The biclustering algorithms operate on both the set of row vectors $X = \{\boldsymbol{x}_1, \ldots \boldsymbol{x}_n\}$ and the set of column vectors $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_d\}$ of data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where n is the number of data points and d the number

of attributes in $\boldsymbol{X}$. Since $x_{p,q} \in \boldsymbol{X}$ is element on row vector $\boldsymbol{x}_p \in X$ and column vector $\boldsymbol{y}_q \in Y$, where $p \in \{1, \ldots, n\}$ and $q \in \{1, \ldots, d\}$, matrix $\boldsymbol{X}$ can be denoted by a tuple (X, Y). In general, biclustering algorithms try to find a set of submatrices $\{(I_1, J_1), \ldots, (I_k, J_k)\}$ of $\boldsymbol{X}$, where $I_i \subseteq X$, $J_i \subseteq Y$, $i \in \{1, \ldots, k\}$ and $k \in \mathbb{N}$. Each of the k submatrices, also know as *biclusters* must satisfy a given homogeneity criterion [37, p. 21]. There are four types of biclusters formed by the pattern-based clustering algorithms: *constant biclusters*, *biclusters with constant values in rows or columns*, *biclusters with coherent values*, and *biclusters with coherent evolutions* [41].

2.3.3 Clustering in Arbitrarily Oriented Subspaces

Even though the processes of finding patterns in biclustering algorithms are easy to interpret, they tend to find patterns which are quite special or even artificial. The main problem of biclustering is caused by Problem 1: The patterns produced by biclustering algorithms often rely on simple mathematical models and cannot represent complex relationships between data points. Contrary to axis-parallel approaches, such as biclustering and axis-parallel subspace clustering, the *Correlation Clustering* algorithms assume that the clusters in data appear in arbitrarily oriented subspaces of $\mathbb{R}^d$, where d denotes the number of dimensions. Correlation clustering algorithms are able to find clusters in which data points have strong linear relationships: The data points can be located along a line or a hyperplane both of which are not axis-parallel. The approaches of correlation clustering differ from each other by the methods they use to determine the orientation of subspaces. [37, p. 31–33]

A large number of existing correlation clustering algorithms use PCA for subspace projection [37, p. 35]. The mathematical background of PCA has been studied from the 19$^{\text{th}}$ century and the modern formalization of PCA was introduced by Hotelling in 1933 [29, 56]. PCA is a well-known statistical multivariate method for extracting important features from data by simplifying data representation and describing the structure of multivariate data [2]. Let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ be an arbitrary data matrix defined in Equation (2.2). The *sample covariance* of vectors $\boldsymbol{x}_i, \boldsymbol{x}_j \in \operatorname{col}(\boldsymbol{X})$, where $\operatorname{col}(\cdot)$ denotes the set of column vectors of a matrix, is defined as

$$\sigma_{i,j} = \operatorname{cov}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{n-1} \langle (\boldsymbol{x}_i - \mu_i \mathbf{1}_n), (\boldsymbol{x}_j - \mu_j \mathbf{1}_n) \rangle, \tag{2.8}$$

where μ_i and μ_j are the mean values of i^{th} and j^{th} vectors of $\operatorname{col}(\boldsymbol{X})$, $i, j \in \{1, 2, \ldots, n\}$ and $\mathbf{1}_n \in \mathbb{R}^n$ is a vector of ones [67, p. 998]. The sample covariance of

vector $\boldsymbol{x}_i$ with itself is called *sample variance* which can be represented as

$$\sigma_i^2 = \operatorname{var}(\boldsymbol{x}_i) = \operatorname{cov}(\boldsymbol{x}_i, \boldsymbol{x}_i) = \frac{1}{n-1}\langle(\boldsymbol{x}_i - \mu_i \mathbf{1}_n), (\boldsymbol{x}_i - \mu_i \mathbf{1}_n)\rangle = \frac{1}{n-1}\|\boldsymbol{x}_i - \mu_i \mathbf{1}_n\|_2^2,$$

where μ_i is the mean value of $\boldsymbol{x}_i$ and $i \in \{1, 2, \ldots, n\}$ [67, p. 987]. PCA starts by computing the *sample covariance* of matrix $\boldsymbol{X}$ whose columns are considered as random variables,

$$\operatorname{cov}(\boldsymbol{X}) = \frac{1}{n-1}\boldsymbol{X}_{\boldsymbol{C}}^T \boldsymbol{X}_{\boldsymbol{C}} = \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,d} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d,1} & \sigma_{d,2} & \cdots & \sigma_d^2 \end{bmatrix} = \boldsymbol{\Sigma}_{\boldsymbol{X}}. \tag{2.9}$$

In Equation (2.9), *centered* data matrix $\boldsymbol{X}_C$ is defined as

$$\boldsymbol{X}_{\boldsymbol{C}} = \boldsymbol{X} - \mathbf{1}_n \boldsymbol{\mu}^T, \tag{2.10}$$

where $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \cdots & \mu_d \end{bmatrix}^T$ is a vector containing mean values of each random variable. Covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{X}}$ of Equation (2.9) has the variances of each vector of $\operatorname{col}(\boldsymbol{X})$ as its diagonal elements and the covariances of all vectors in its other elements. After computing the covariance matrix of $\boldsymbol{X}$, PCA proceeds by solving the *eigenvalues* $\lambda \in \mathbb{R}_+$ which are the roots of the equation

$$\det(\boldsymbol{\Sigma}_{\boldsymbol{X}} - \lambda \boldsymbol{I}) = 0, \tag{2.11}$$

where $\det(\cdot)$ denotes the *determinant* of a matrix and $\boldsymbol{I}$ denotes the *identity matrix*. Equation (2.11) is called the *characteristic equation* [57, p. 235]. Thus, for each distinct eigenvalue λ there exists an *eigenvector* $\boldsymbol{v} \in \mathbb{R}^n$ such that

$$\boldsymbol{\Sigma}_{\boldsymbol{X}} \boldsymbol{v} = \lambda \boldsymbol{v}. \tag{2.12}$$

Equation (2.12) is called the *eigenvalue equation*. If the roots $\lambda \in \mathbb{R}_+$ of Equation (2.11) are distinct, eigenvectors of matrix $\boldsymbol{\Sigma}_{\boldsymbol{X}}$ *diagonalize* the matrix:

$$\boldsymbol{\Sigma}_{\boldsymbol{X}} = \boldsymbol{V} \boldsymbol{\Lambda} \boldsymbol{V}^{-1}, \tag{2.13}$$

where $\boldsymbol{V}$ is the *eigenvector matrix*, $\boldsymbol{V}^{-1}$ denotes the inverse of matrix $\boldsymbol{V}$, and $\boldsymbol{\Lambda}$ is the *eigenvalue matrix* which has the eigenvalues as its diagonal in a decreasing order [57, p. 245]. The vectors of $\boldsymbol{V}$ in Equation (2.13) are mutually orthogonal [38, p. 576]. Matrix $\boldsymbol{\Lambda}$ in Equation (2.13) can be taught as the presentation of covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{X}}$ in a different coordinate system where the first eigenvector $\boldsymbol{v}_1 \in \operatorname{col}(\boldsymbol{V})$ points

to the direction of highest variance in $\boldsymbol{X}$ [37, p. 34]. The PCA-based correlation clustering algorithms differ from each other in the way they determine the most interesting subspaces genereated by PCA and in the similarity measures they use.

The algorithm, proposed as the first *generalized projected clustering* by Aggarwal and Yu [6], called *ORCLUS* determines the desired number of clusters k, as in k-means, from the input data. ORCLUS proceeds iteratively reducing the number of initial clusters from k_0 to desired k by merging existing clusters and reducing their full dimensionality from l_0 to desired l according to predefined factors. ORCLUS proceeds according to the following steps:

1. ORCLUS starts by selecting $k_c = k_0 > k$ *seeds* $s_1, \ldots, s_{k_c}$ from the original data set and assigning all the data points to the closest clusters $C_1, \ldots, C_{k_c}$ represented by the seeds according to the chosen distance function in each cluster's current *eigensystem* $\mathcal{E}_i \in \{\mathcal{E}_1, \ldots, \mathcal{E}_{k_c}\}$. Each eigensystem is a set of vectors representing the current projection of coordinate axes which is initially the original coordinate system for all eigensystems $\mathcal{E}_i \in \{\mathcal{E}_1, \ldots, \mathcal{E}_{k_c}\}$.

2. Each cluster's C_i centroid is computed according to the distance function *dist* and assigned as the cluster's new seed s_i. Similarly as in PCA, the orthogonal basis of each cluster C_i is computed and set as the new eigensystem $\mathcal{E}_i$ of the cluster. As ORCLUS reduces the dimensionality of cluster eigensystems through its iterations, only the l_c vectors corresponding the l_c smallest eigenvalues are selected as the cluster's eigensystem. When the eigensystems are computed for the initial $k_c = k_0$ clusters, l_0 is used as the number of vectors in an eigensystem.

3. Throughout every iteration of ORCLUS, the number of clusters is reduced from the initial number of k_0 clusters. The number of clusters and dimensionality of data is reduced according to predefined factors $\alpha < 1$ and $\beta < 1$ for which

$$\log_{1/\alpha}\left(\frac{k_0}{k}\right) = \log_{1/\beta}\left(\frac{l_0}{l}\right),$$

   where α is the factor reducing the number of clusters and β the factor reducing the dimensionality of each cluster. The larger the values of α and β are, the less iterations are performed in a process, where k_0 clusters are merged to k resulting clusters.

4. The number of clusters is reduced by merging the current clusters: Two clusters with the smallest *projected energy* in their joint eigensystem are combined. The

projected energy of a cluster is the mean value of squared distances between each data point and the cluster centroid in the cluster's eigensystem:

$$R(C, \mathcal{E}) = \frac{1}{n} \sum_{i=1}^{n} Pdist(\boldsymbol{x}_i, \boldsymbol{m}_C, \mathcal{E})^2, \tag{2.14}$$

where C denotes an arbitrary cluster, $\mathcal{E}$ the eigensystem of C, m_C the centroid of C, and $Pdist(\boldsymbol{x}_i, \boldsymbol{m}_C, \mathcal{E})$ the *projected distance* between a data point x_i and $\boldsymbol{m}_C$ in the eigensystem of C. The projected distance $Pdist$ the same distance function as *dist* mentioned in Step 2 of ORCLUS but it is computed in a chosen eigensystem. The cluster merging continues until the number of clusters has decreased by the factor of α.

5. Steps 2–4 will continue until $k_c = k$.

Even though ORCLUS is able to find correlating clusters in arbitrarily oriented subspaces by eliminating the most sparse subspaces, it is computationally quite expensive and it does not provide any clear mathematical structure of the resulting clusters. In addition, the algorithm's optimal hyperparameter setting can be hard to find which makes the usage of ORCLUS unintuitive. [6]

Even though PCA-based correlation clustering algorithms can detect correlation clusters in arbitrarily oriented subspaces in a mathematically expressive way, they often collide with Problem 2. A general method for improving PCA-based algorithms has been proposed by Kriegel et al. [36]: Before applying PCA on data, each data point is assigned to a correct *correlation neighborhood* based dense regions in data. PCA is applied to each correlation neighborhood separately. Since the method relies on dense regions in data, it does not solve Problem 2 in truly high dimensions. It is evident that study of PCA-based algorithms needs to focus on overcoming Problem 2 in the future.

Some approaches in correlation clustering use *Hough Transform* [3] for finding the arbitrarily oriented subspaces in multidimensional spaces. The Hough transform was originally designed to map data points from *picture space* to *parameter space* for detecting complex patterns in data [30]. Since the mapped data points are represented as trigonometric functions in parameter space, the locality assumption is totally dismissed [37, p. 39]. Thus, the challenges related to Problem 2 are not present. The recent study of clustering algorithms using Hough transform [32] has produced impressive results on detecting clusters from data with non-linear correlation relationships. Other interesting approaches for finding clusters in arbitrarily oriented subspaces in high dimensions include the use of *fractal dimensions* [25] for

overcoming Problem 3 and *random sampling* [27] for lowering the computational costs.

As the curse of dimensionality proposes several unexpected problems in clustering high-dimensional data, the large number of existing algorithms designed for this purpose often solve problems within a certain application. Fortunately, the study of correlation clustering algorithms seems to be heading towards a direction of generalized and problem-oriented solutions. Especially, the PCA-based algorithms have succeeded in producing mathematically expressive clusters and hierarchies between them. Since the matrix diagonalization itself is a powerful technique for getting information about the internal structures of a matrix, it provides endless possibilities in terms of clustering – the full potential of PCA-based correlation algorithms has not been reached yet. This thesis proposes two new PCA-based algorithms which hierarchically cluster high-dimensional based on the variations of the most significant eigenvectors of each data point. The design of these algorithms relies on mathematically expressive result presentation and intuitive hyperparameter setting.

3. ALGORITHMS FOR CLUSTERING HIGH-DIMENSIONAL DATA

This chapter introduces two new PCA-based correlation clustering algorithms CHUNX and CRUSHES. From a problem-oriented view of clustering high-dimensional data, PCA is a sophisticated method to overcome several issues related to Problem 1 and therefore it is suitable as the basis of CHUNX and CRUSHES. The main objective of CHUNX and CRUSHES is to provide not only easy-to-use and mathematically expressive clustering algorithms for applications of high-dimensional data but generic tools for exploring the internal structures of data exposed by matrix diagonalization. These algorithms are designed to be used on their own or as an extended version of PCA.

3.1 Illustration of CHUNX and CRUSHES

CHUNX and CRUSHES cluster high-dimensional data hierarchically into clusters based on the angles between single data points $\boldsymbol{x} \in \boldsymbol{X}$ and vectors in the orthogonal basis $\boldsymbol{V}$ of $\boldsymbol{X}$. The angles between the data points and each vector in $\boldsymbol{V}$ are labeled as $\pm\theta_i$, where $i = 1, \ldots, d$ and $0 \leq \theta_i < \frac{\pi}{2}$. Since the vectors in $\boldsymbol{V}$ are mutually orthogonal, it is essential to take both the positive and negative vector directions into account. Thus, each $\boldsymbol{x}$ can be labeled as permutations of component labels, where the labels are in an increasing order by the size of each angle between $\boldsymbol{x}$ and vector in $\boldsymbol{V}$. The vector of $\boldsymbol{V}$ which has the smallest angle between $\boldsymbol{x}$ is referred to as *the most significant component* of $\boldsymbol{x}$

In principle, CHUNX iterates through each formed component label permutation starting from the component which has the smallest angle with each $\boldsymbol{x} \in \boldsymbol{X}$. The data points in $\boldsymbol{X}$ with the same most significant components form a set of clusters $\{C_1, \ldots, C_k\}$, where k is the number of unique most significant components. Thus, the first level of clustering hierarchy is formed. If the data point count in $C_i \in \{C_1, \ldots, C_k\}$ is below the value of hyperparameter *maximum cluster size* $n_{\max}$, C_i remains as a cluster. Otherwise CHUNX proceeds recursively by clustering each C_i by the next most significant components of data points in the subclusters of each

C_i. CHUNX continues until each cluster satisfies the *maximum size criterion*. The resulting CHUNX clusters labeled as variations of component labels. Even though the name CHUNX is an abbreviation, it has its etymology in the word "chunk", since CHUNX is a straightforward process of splitting a blob of data into chunks of different sizes and shapes.

Unlike CHUNX, which clusters $\boldsymbol{X}$ hierarchically into clusters with a size less than $n_{\max}$, CRUSHES produces clusters in which the data points correlate mutually at certain level. CRUSHES aims at representing each $\boldsymbol{x} \in \boldsymbol{X}$ with a minimum number of most significant components. The minimum number of components is determined by hyperparameter *mutual correlation* ρ – a representation of a data point must have a correlation greater than ρ with its original data point. As in CHUNX, the cluster labels in CRUSHES are variations of component labels. The name CRUSHES refers to crushing which produces crumbs – a large number of tiny ones but some significantly larger as well.

Using the component labeling information, both algorithms construct a *tree* data structure in which the *root node* represents the entire data set. Figure 3.1 illustrates the principle structure of a tree resulting from CHUNX and CRUSHES. Each *child*

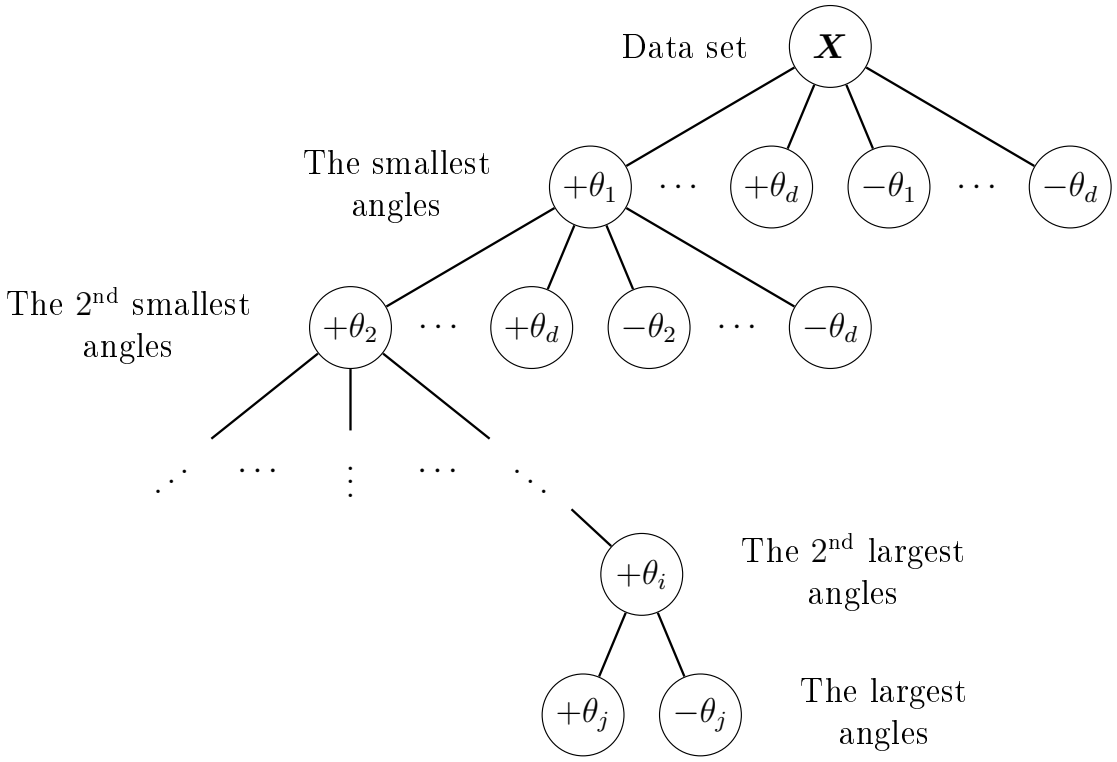

Figure 3.1 *The principal tree structure formed by CHUNX and CRUSHES.*

node of the root represent the set of sample vectors which have the smallest angle with the same vector in $\boldsymbol{V}$ or in $-\boldsymbol{V}$. There are at most $2d$ child nodes for the root.

As the *depth* of the tree increases, the maximum number of possible child nodes for each node reduces by two, since both the parent node's vector and its opposite vector are reduced from the number of possible child nodes. The maximum number of nodes in a tree with depth of D can be expressed as the sum of angle label variations,

$$\begin{aligned} n_{\text{max nodes}}(D) &= 1 + 2d + 2d(2d-2) + \cdots + 2d(2d-2)\cdots(2d-2D) \\ &= 2^0 + 2^1 d + 2^2 d(d-1) + \cdots + 2^D d(d-1)\cdots(d-D) \\ &= 2^0 \frac{d!}{(d-0)!} + 2^1 \frac{d!}{(d-1)!} + 2^2 \frac{d!}{(d-2)!} + \ldots + 2^D \frac{d!}{(d-D)!} \\ &= \sum_{i=0}^{D} 2^i \frac{d!}{(d-i)!}, \end{aligned} \tag{3.1}$$

where $0 \leq D \leq d$ and ! denotes the *factorial* of an integer. It should be noted that the maximum number of nodes in a tree with any depth would only appear for uniformly distributed spherical data over d dimensions which obviously does not cluster well. Each cluster label in CHUNX and CRUSHES is a *path* – a sequence of node labels – between the root and a *descendant node*.

The difference between CHUNX and CRUSHES is clearly visible in the structures of the trees they construct – in a CHUNX tree the clusters appear only in its leaf nodes which is not a necessary for a CRUSHES tree. An imaginary example of a CHUNX tree is illustrated in Figure 3.2 in which a data matrix $\boldsymbol{X} \in \mathbb{R}^{500 \times d}$, where $d > 5$, is clustered into 8 clusters with $n_{\max} = 100$. The leaf nodes with grey fill and

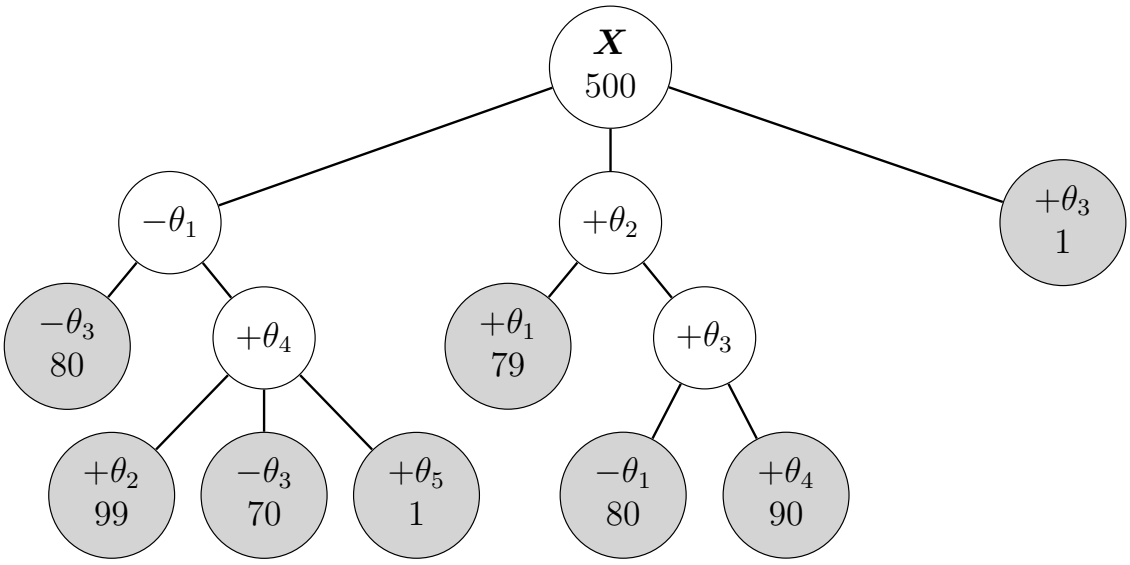

Figure 3.2 *An imaginary example of CHUNX clustering hierarchy with $n = 500$ and $n_{max} = 100$. The number in each node represents the data point count. The leaf nodes marked with grey fill represent the clusters.*

element counts represent the formed clusters in Figure 3.2, where the labels of the 8 clusters are: $[-\theta_1, -\theta_3]$, $[-\theta_1, +\theta_4, +\theta_2]$, $[-\theta_1, +\theta_4, -\theta_3]$, $[-\theta_1, +\theta_4, +\theta_5]$, $[+\theta_2, +\theta_1]$, $[+\theta_2, +\theta_3, -\theta_1]$, $[+\theta_2, +\theta_3, +\theta_4]$, and $[+\theta_3]$. For comparison, Figure 3.3 illustrates an imaginary example of a CRUSHES tree, where a data matrix $\boldsymbol{X}$ with similar properties as in 3.2 is clustered into 8 clusters with some value of ρ. The nodes with

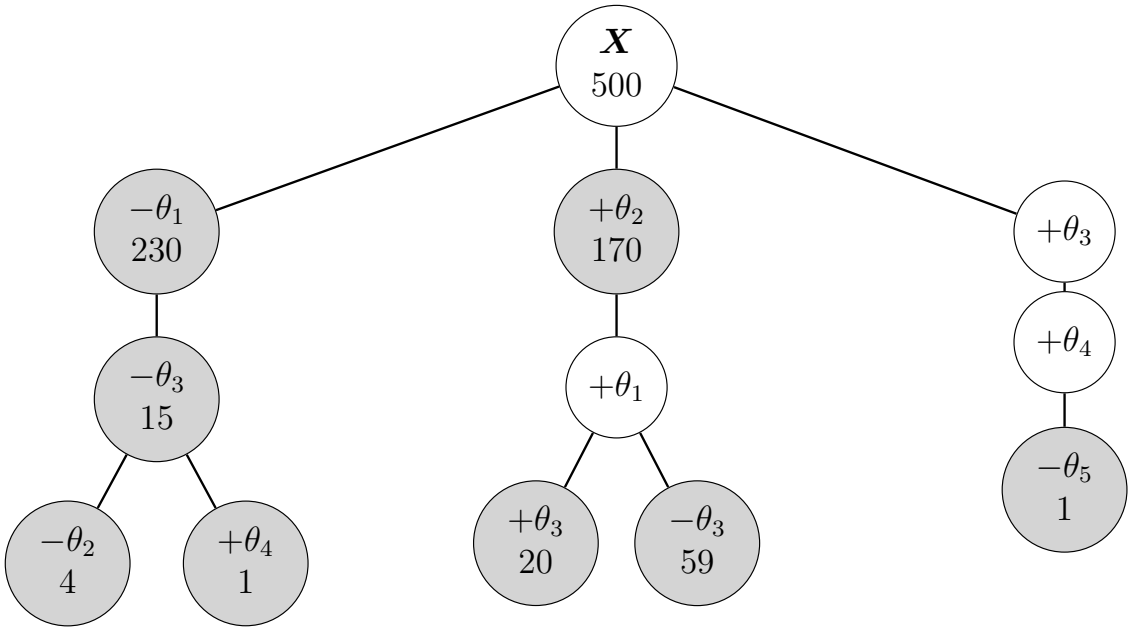

Figure 3.3 *An imaginary example of CRUSHES clustering hierarchy with $n = 500$, where the number in each node represents the element count. The nodes with grey fill of the tree represent the clusters.*

grey fill and element counts in Figure 3.3 represent the clusters with labels $[-\theta_1]$, $[-\theta_1, -\theta_3]$, $[-\theta_1, -\theta_3, -\theta_2]$, $[-\theta_1, -\theta_3, +\theta_4]$, $[+\theta_2]$, $[+\theta_2, +\theta_1, +\theta_3]$, $[+\theta_2, +\theta_1, -\theta_3]$, and $[+\theta_3, +\theta_4, -\theta_5]$. Unlike the CHUNX clusters, the CRUSHES clusters are not necessarily leaf nodes. It should be noted that in Figure 3.3 the cluster counts of nodes which are not leaves represent the data points which belong only to that particular cluster – the clusters appearing in the child nodes of those clusters are not included in the ancestor node's element count. Hence the cluster counts sum up to the total number of instances which is 500 in the above example.

The trees representing the clustering hierarchy of CHUNX and CRUSHES provide a mathematically expressive illustration of the complex relationships within the data matrix and they provide a data structure for noise reduction: As a tree is constructed, it can be filtered with queries related to certain component variations and cluster sizes. Since both CHUNX and CRUSHES build their tree hierarchies based on the most significant components of data, they are not sensitive for Problem 3. Contrary to many PCA-based algorithms, CHUNX and CRUSHES avoid Problem 4 by not dismissing any component of the data sets orthogonal basis $\boldsymbol{V}$. The ways to

overcome the issues related to Problem 2 are discussed in the next section in which the mathematical background of CHUNX and CRUSHES is presented in detail.

3.2 Mathematical Background

This section describes mathematical features of CHUNX and CRUSHES algorithms and workarounds for Problem 2 which is a common issue in most correlation clustering algorithms. Since the main features of PCA are already described in subsection 2.3.3, this section focuses on mathematical constructions derived from it.

Both CHUNX and CRUSHES operate on data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ similar to (2.2). Since both algorithms are designed for clustering shapes of multidimensional data series, $\boldsymbol{X}$ is centered and its data vectors are *normalized*. Each data point $\boldsymbol{x} \in \text{row}\,(\boldsymbol{X})$ is centered around *origin* by removing the mean values of each column of $\boldsymbol{X}$ from each data point $\boldsymbol{x}$ similarly as in Equation (2.10). Data points are normalized by their vector length,

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2},$$

where $\|\boldsymbol{x}\|_2$ is the Euclidean norm of $\boldsymbol{x}$ described in (2.3). Thus, $\hat{\boldsymbol{X}}_c$ denotes the centered and normalized data matrix. In order to clarify the mathematical notations,

$$\boldsymbol{X} = \hat{\boldsymbol{X}}_c \tag{3.2}$$

in further mathematical expressions.

CHUNX and CRUSHES base their mathematical inference on the orthogonal basis $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d\}$ for $\mathbb{R}^d$ which is computed for the covariance matrix $\boldsymbol{\Sigma_X}$ of matrix $\boldsymbol{X}$ using the methods of PCA described in section 2.3.3. In order to compute the angle magnitudes between each data point and each projected coordinate axis, it must be shown that the set $\{\boldsymbol{X}\boldsymbol{v}_1, \boldsymbol{X}\boldsymbol{v}_2, \ldots, \boldsymbol{X}\boldsymbol{v}_d\}$ is orthogonal as well.

The set $\{\boldsymbol{X}\boldsymbol{v}_1, \boldsymbol{X}\boldsymbol{v}_2, \ldots, \boldsymbol{X}\boldsymbol{v}_d\}$ is orthogonal if

$$\langle \boldsymbol{X}\boldsymbol{v}_i, \boldsymbol{X}\boldsymbol{v}_j \rangle = 0,$$

for all i and j, when $i \neq j$ and $i, j \in \{1, 2, \ldots, d\}$. Since $\boldsymbol{X}$ is centered, by Equation (2.9),

$$\boldsymbol{\Sigma_X} = \frac{1}{n-1}\boldsymbol{X_C}^T\boldsymbol{X_C} = \frac{1}{n-1}(\boldsymbol{X} - \boldsymbol{0})^T(\boldsymbol{X} - \boldsymbol{0}) = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X}, \tag{3.3}$$

where $\mathbf{0} \in \mathbb{R}^{n \times d}$ denotes a matrix with only zero elements. For arbitrary eigenvectors $\boldsymbol{v}_i, \boldsymbol{v}_j \in \operatorname{col}(\boldsymbol{V})$, where $i \neq j$ and $i, j \in \{1, 2, \ldots, d\}$, by Equations (2.12) and (3.3)

$$\begin{aligned}\langle \boldsymbol{X}\boldsymbol{v}_i, \boldsymbol{X}\boldsymbol{v}_j \rangle &= \left(\boldsymbol{X}\boldsymbol{v}_i\right)^T \boldsymbol{X}\boldsymbol{v}_j = \boldsymbol{v}_i^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{v}_j = \boldsymbol{v}_i^T (n-1)\boldsymbol{\Sigma}_{\boldsymbol{X}} \boldsymbol{v}_j = \boldsymbol{v}_i^T (n-1)\lambda_j \boldsymbol{v}_j \\ &= (n-1)\lambda_j \boldsymbol{v}_i^T \boldsymbol{v}_j = (n-1)\lambda_j \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = 0,\end{aligned}$$

since the eigenvectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are orthogonal. Thus the set $\{\boldsymbol{X}\boldsymbol{v}_1, \boldsymbol{X}\boldsymbol{v}_2, \ldots, \boldsymbol{X}\boldsymbol{v}_n\}$ is orthogonal.

The matrix forming the orthogonal basis of $\boldsymbol{X}$,

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_d \end{bmatrix} \in \mathbb{R}^{d \times d}, \tag{3.4}$$

can be computed by diagonalizing matrix $\boldsymbol{\Sigma}_{\boldsymbol{X}}$ as given in Equation (2.13). Since the vectors in $\boldsymbol{V}$ are orthogonal, each data point $\boldsymbol{x}_i \in \operatorname{row}(\boldsymbol{X})$ can be represented as a linear combination of every vector $\boldsymbol{v}_j \in \operatorname{col}(\boldsymbol{V})$:

$$\begin{aligned}\langle \boldsymbol{x}_i, \boldsymbol{v}_j \rangle &= \langle a_{i,1}\boldsymbol{v}_1 + a_{i,2}\boldsymbol{v}_2 + \cdots + a_{i,d}\boldsymbol{v}_d, \boldsymbol{v}_j \rangle \\ &= \langle a_{i,1}\boldsymbol{v}_1, \boldsymbol{v}_j \rangle + \langle a_{i,2}\boldsymbol{v}_2, \boldsymbol{v}_j \rangle + \cdots + \langle a_{i,d}\boldsymbol{v}_d, \boldsymbol{v}_j \rangle \\ &= a_{i,1}\langle \boldsymbol{v}_1, \boldsymbol{v}_j \rangle + a_{i,2}\langle \boldsymbol{v}_2, \boldsymbol{v}_j \rangle + \cdots + a_{i,d}\langle \boldsymbol{v}_d, \boldsymbol{v}_j \rangle \\ &= 0 + \cdots + 0 + a_{i,j}\langle \boldsymbol{v}_j, \boldsymbol{v}_j \rangle + 0 + \cdots + 0 \\ &= a_{i,j},\end{aligned} \tag{3.5}$$

where $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, d\}$. Since the lengths of vectors in $\operatorname{row}(\boldsymbol{X})$ are normalized, coefficient $a_{i,j}$ corresponds to

$$a_{i,j} = \|\boldsymbol{x}_i\|_2 \|\boldsymbol{v}_j\|_2 \cos(\theta_{i,j}) = \frac{\|\boldsymbol{x}_i\|_2 \|\boldsymbol{v}_j\|_2 \cos(\theta_{i,j})}{\|\boldsymbol{x}_i\|_2 \|\boldsymbol{v}_j\|_2} = \frac{\langle \boldsymbol{x}_i, \boldsymbol{v}_j \rangle}{\|\boldsymbol{x}_i\|_2 \|\boldsymbol{v}_j\|_2} = \operatorname{corr}(\boldsymbol{x}_i, \boldsymbol{v}_j),$$

where $\theta_{i,j}$ is the angle between $\boldsymbol{x}_i$ and $\boldsymbol{v}_j$. Correlation represents well the angle magnitude between two vectors. Correlations between each data point $\boldsymbol{x} \in \operatorname{row}(\boldsymbol{X})$ and each positive eigenvector $\boldsymbol{v} \in \operatorname{col}(\boldsymbol{V})$, *factor loads* of $\boldsymbol{X}$, can be computed from the matrix product

$$\boldsymbol{P} = \boldsymbol{X}\boldsymbol{V} \tag{3.6}$$

and the angles between each $\boldsymbol{x} \in \operatorname{row}(\boldsymbol{X})$ and each positive eigenvector $\boldsymbol{v} \in \operatorname{col}(\boldsymbol{V})$ from

$$\boldsymbol{\Theta}_+ = \arccos(\boldsymbol{P}). \tag{3.7}$$

Similarly, the angles between the negative eigenvectors can be computed from

$$\boldsymbol{\Theta}_- = \arccos(-\boldsymbol{P}). \tag{3.8}$$

In order to take into account both the positive and negative signs of each eigenvector $\boldsymbol{v} \in \operatorname{col}(\boldsymbol{V})$, matrix

$$\boldsymbol{\Theta}_{\pm} = \begin{bmatrix} \boldsymbol{\Theta}_{+} & \boldsymbol{\Theta}_{-} \end{bmatrix} \tag{3.9}$$

can be used.

The mathematical constructions required for CHUNX algorithm are presented in Equations (3.3), (3.4), (3.6), (3.7), and (3.9). Beyond the constructions of CHUNX, CRUSHES needs a way to investigate the proportions of each $\boldsymbol{v} \in \operatorname{col}(\boldsymbol{V})$ explaining the variance of each $\boldsymbol{x} \in \operatorname{row}(\boldsymbol{X})$. Let $\boldsymbol{x} \in \boldsymbol{X}$ be an arbitrary vector, $\boldsymbol{V}$ the matrix forming the orthogonal basis of $\boldsymbol{X}$, and

$$\boldsymbol{x}' = \sum_{i \in \boldsymbol{k}} a_i \boldsymbol{v}_i,$$

where $\boldsymbol{k} \in \mathbb{R}^k$ is a vector of indices of k most significant components of $\boldsymbol{x}$ and $1 \leq k \leq d$. Correlation between $\boldsymbol{x}$ and $\boldsymbol{x}'$ can be expressed as

$$\begin{aligned} \operatorname{corr}(\boldsymbol{x}, \boldsymbol{x}') &= \frac{\langle \boldsymbol{x}, \boldsymbol{x}' \rangle}{\|\boldsymbol{x}\|_2 \|\boldsymbol{x}'\|_2} = \frac{1}{\|\boldsymbol{x}'\|_2} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle = \frac{1}{\|\boldsymbol{x}'\|_2} \left\langle \boldsymbol{x}, \sum_{i \in \boldsymbol{k}} a_i \boldsymbol{v}_i \right\rangle \\ &= \frac{1}{\|\boldsymbol{x}'\|_2} \left(\langle \boldsymbol{x}, a_{k_1} \boldsymbol{v}_{k_1} \rangle + \cdots + \langle \boldsymbol{x}, a_{k_k} \boldsymbol{v}_{k_k} \rangle \right) = \frac{1}{\|\boldsymbol{x}'\|_2} \sum_{j \in \boldsymbol{k}} \langle \boldsymbol{x}, a_j \boldsymbol{v}_j \rangle \\ &= \frac{1}{\|\boldsymbol{x}'\|_2} \sum_{j \in \boldsymbol{k}} a_j \langle \boldsymbol{x}, \boldsymbol{v}_j \rangle . \end{aligned} \tag{3.10}$$

By Equation (3.5),

$$\operatorname{corr}(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{\|\boldsymbol{x}'\|_2} \sum_{j \in \boldsymbol{k}} a_j \langle \boldsymbol{x}, \boldsymbol{v}_j \rangle = \frac{1}{\|\boldsymbol{x}'\|_2} \sum_{j \in \boldsymbol{k}} a_j^2, \tag{3.11}$$

hence the correlation of $\boldsymbol{x}$ and $\boldsymbol{x}'$ is decomposed as a sum in which each element represents the proportion each $\boldsymbol{v} \in \operatorname{col}(\boldsymbol{V})$ explain from the variance of $\boldsymbol{x}$. In order to compute the proportions of explained variance for each $\boldsymbol{v} \in \operatorname{col}(\boldsymbol{V})$ over the whole data $\operatorname{row}(\boldsymbol{X})$, one must simply compute

$$\boldsymbol{P}^2 = \begin{bmatrix} a_{1,1}^2 & \cdots & a_{1,d}^2 \\ \vdots & \ddots & \vdots \\ a_{n,1}^2 & \cdots & a_{n,d}^2 \end{bmatrix}, \tag{3.12}$$

where $\boldsymbol{P}$ is the factor load matrix of Equation (3.6) and the matrix squaring operator denotes element-wise squaring.

Since both CHUNX and CRUSHES operate on centered and normalized data, they dismiss the clustering by data scale. As CHUNX and CRUSHES focus on approximate data vector directions by grouping data points by the angles between them and their most significant components, they operate very loosely on Euclidean distances between data points. In other words, CHUNX and CRUSHES cluster shapes of high-dimensional data. Therefore, they practically avoid Problem 2.

3.3 CHUNX Algorithm

CHUNX is a correlation clustering algorithm for high-dimensional data which clusters the input data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ recursively into clusters by the ascending order of angles between an individual data point and each vector in the orthogonal basis of $\boldsymbol{X}$. CHUNX assumes that the data matrix $\boldsymbol{X}$ is represented as the one in Equation (2.2) – data points are d-dimensional vectors $\boldsymbol{x} \in \mathrm{col}\,(\boldsymbol{X})$. It is also required that Equation (3.2) holds. Algorithm 1 presents CHUNX in pseudocode and uses the symbols of mathematical constructions defined in Section 3.2 in its presentation.

Algorithm 1 CHUNX algorithm

```
1  function CHUNX(X, p_max, c_max)
2      inputs: X,     Centered R^{n×d} matrix, where n > d, with data points as
3                     length normalized rows and attributes as columns.
4              p_max, Relative maximum cluster size
5              c_max, Maximum depth of recursion. (Optional)
6      returns the list of cluster labels for rows of input matrix X
7
8      Σ_X ← cov(X)                                  ▷ Covariance matrix of X
9      V ← eigenvectors(Σ_X)                         ▷ The orthogonal basis
10     P ← XV                                        ▷ Factor loads of X
11     Θ_+ ← arccos(P)                               ▷ Angles with positive eigenvectors
12     Θ_- ← arccos(−P)                              ▷ Angles with negative eigenvectors
13     Θ_± ← concatenate(Θ_+, Θ_-)
14     L' ← argsort(Θ_±, row-wise)
15     L ← L'[:, range(n/2)]                         ▷ Only the first n/2 columns are needed
16     n_max ← ceil(p_max n)                         ▷ Closest natural number with ceiling function
17     return CHUNX-PARTITION(L, n_max, c_max)
18 end function
```

In addition to $\boldsymbol{X}$, CHUNX takes two hyperparameters *relative maximum cluster size* $p_{\max}$ and *maximum depth of recursion* which is an optional parameter. On the line 16 of Algorithm 1, the product of hyperparameter $p_{\max} \in (0, 1)$ and the number of data points n rounded up to closest integer sets the maximum cluster size $n_{\max}$ which has been mentioned in Section 3.1 on page 19. CHUNX returns a list of cluster labels

each of which are variations of angle labels. The format of cluster labels has been explained in Section 3.1 starting from page 21.

CHUNX starts by mapping each data point $\boldsymbol{x} \in \mathrm{col}\,(\boldsymbol{X})$ into an angle label permutation which are used in the actual cluster formation. In order to generate the permutations, CHUNX computes the orthogonal basis $\boldsymbol{V}$ of $\boldsymbol{X}$, defined in Equation (3.4). This is done by computing $\boldsymbol{V}$ for the covariance matrix $\boldsymbol{\Sigma_X}$ of $\boldsymbol{X}$ defined in Equation (3.3). The computations required for evaluation of $\boldsymbol{V}$ are presented on lines 8 and 9 of Algorithm 1. Starting from line 10 of Algorithm 1, the factor loads of each data point are computed and transformed into angles between a data point and each vector of $\mathrm{col}\,(\boldsymbol{V})$ as in Equations (3.6) and (3.7).

In order to get angles that are in $[0, \pi)$ for both positive and negative directions of each eigenvector, a matrix $\boldsymbol{\Theta}_{\pm}$ of Equation (3.9) is formed on line 13 of Algorithm 1. The indices of positive and negative eigenvectors of each row in matrix $\boldsymbol{\Theta}_{\pm}$ are sorted into an ascending order $\boldsymbol{L'} \in \mathbb{R}^{n \times 2d}$ by their angle magnitudes. The eigenvectors with a negative sign have indices in the range from $d + 1$ to $2d$. On line 15 of Algorithm 1, the last d columns of $\boldsymbol{L'}$ are discarded, since they are complementary angles of the first n elements in a reverse order an not in $[0, \pi)$.

After each data point has been mapped as a permutation of eigenvector labels, the matrix $\boldsymbol{L}$ is passed to CHUNX-PARTITION function in which it is referred as $\boldsymbol{A}$. CHUNX-PARTITION clusters $\boldsymbol{X}$ recursively based on $\boldsymbol{A}$ and hyperparameters of CHUNX $n_{\max}$ and $c_{\max}$. Algorithm 2 presents CHUNX-PARTITION in pseudocode. It starts by initializing an empty array of eigenvector label variations. Before every run, on lines 10–15 of Algorithm 2, the *optional* hyperparameter $c_{\max}$ is decremented and checked – if the maximum number of labels in a variation is exceeded, the recursion terminates with an empty list of variations.

After the optional check of label count, on line 16 of Algorithm 2, CHUNX selects the first column $\boldsymbol{a}_1$ of $\boldsymbol{A}$, which represents the eigenvectors with the smallest angle between the data points from which the unique elements A_1 and the counts of the unique elements C_1 are computed on lines 17 and 18 of Algorithm 2. The unique elements A_1 determine the first level of clustering hierarchy. In the beginning of for-loop on lines 19–31 of Algorithm 2, the indices of $\boldsymbol{a}_1$, which have the same label $(A_1[j])$, are selected (line 20 of Algorithm 2) and initialized to the list of variations $\boldsymbol{l}$ at their original index in $\boldsymbol{a}_1$ (lines 21–23 of Algorithm 2). After the initialization, on line 24 of Algorithm 2, the current cluster size is compared with hyperparameter $n_{\max}$. If the current cluster size is less or equal to n_{relax}, the recursion will terminate and the eigenvector label variations in the current cluster will be returned. Oth-

Algorithm 2 CHUNX-PARTITION algorithm

```
1  function CHUNX-PARTITION(A, n_max, c_max)
2      inputs: A,     The sorted order of arguments in the original matrix X,
3                     where the arguments correspond to the order of the
4                     eigenvectors of the original data matrix X.
5              n_max, The maximum cluster size
6              c_max, The maximum depth of recursion. (Optional)
7      returns an array of eigenvector label permutations l
8
9      l ← ∅                          ▷ The array of permutations for each data point
10     if c_max ≠ null then                          ▷ Limitation for permutation size
11         c_max ← c_max − 1
12         if c_max < 0 then
13             return l
14         end if
15     end if
16     a_1 ← A[:, 1]          ▷ The most significant components for each element
17     A_1 ← {a | a ∈ a_1}                          ▷ The list of unique elements in a_1
18     C_1 ← {count(a) | a ∈ a_1}                   ▷ The list of element counts in a_1
19     for j ∈ range(size(A_1)) do
20         i ← {index(a) | a ∈ a_1, a = A_1[j]}     ▷ The index list where a = A_1[j]
21         for i ∈ i do
22             l[i[i]] ← {A_1[j]}                          ▷ Initialize the permutation
23         end for
24         if C_1[j] > n_max then                  ▷ Check if cluster is over n_max
25             A' ← A[i, range(2, size(A))]        ▷ Rows in i without a_1 ∈ A
26             l' ← CHUNX-PARTITION(A', n_max, c_max)
27             for k ∈ range(size(i)) do
28                 l[i[k]].append(l'[k])                   ▷ Append the permutation
29             end for
30         end if
31     end for
32     return l
33 end function
```

erwise the recursion will continue by calling CHUNX-PARTITION, on line 26 of Algorithm 2, with the submatrix $\boldsymbol{A'}$ of $\boldsymbol{A}$ where only the rows corresponding the current cluster are selected and the first column of $\boldsymbol{A}$ is omitted. As the CHUNX-PARTITION terminates its recursion, the eigenvector label variations are returned to a temporary list $\boldsymbol{l'}$ whose elements are appended to the resulting list $\boldsymbol{l}$ and assigned to their original indices at $\boldsymbol{l}$ on lines 27–29 of Algorithm 2.

The final list of eigenvector label variations returned by CHUNX-PARTITION on line 17 of Algorithm 1 is the result of CHUNX algorithm. The labels of the clusters can be determined by finding the unique variations from the resulting list. As the

resulting cluster labels are variations of angle labels, a CHUNX tree illustrated in Figure 3.2 can be generated from them.

3.4 CRUSHES Algorithm

CRUSHES is a correlation clustering algorithm for high-dimensional data which clusters the input data $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ into clusters by a mutual correlation estimation set for clusters. The same assumptions as in CHUNX, described in Section 3.3, hold also for CRUSHES: The data points are d-dimensional vectors $\boldsymbol{x} \in \mathrm{col}\,(\boldsymbol{X})$ and Equation (3.2) holds. The pseudocode of CRUSHES is presented in Algorithm 3.

Algorithm 3 CRUSHES algorithm

function CRUSHES($\boldsymbol{X}, \rho$)
  inputs: $\boldsymbol{X}$, Centered $\mathbb{R}^{n \times d}$ matrix, where $n > d$, with data points as length normalized rows and attributes as columns.
  ρ, Mutual correlation within cluster
  returns the list of cluster labels for rows of input matrix $\boldsymbol{X}$

  $\boldsymbol{\Sigma_X} \leftarrow \mathrm{cov}\,(\boldsymbol{X})$ ▷ Covariance matrix of $\boldsymbol{X}$
  $\boldsymbol{V} \leftarrow \mathrm{eigenvectors}\,(\boldsymbol{\Sigma_X})$ ▷ The orthogonal basis
  $\boldsymbol{P} \leftarrow \boldsymbol{XV}$ ▷ Factor loads of $\boldsymbol{X}$
  $\boldsymbol{\Theta}_+ \leftarrow \arccos(\boldsymbol{P})$ ▷ Angles with positive eigenvectors
  $\boldsymbol{\Theta}_- \leftarrow \arccos(-\boldsymbol{P})$ ▷ Angles with negative eigenvectors
  $\boldsymbol{\Theta}_\pm \leftarrow \mathrm{concatenate}\,(\boldsymbol{\Theta}_+, \boldsymbol{\Theta}_-)$
  $\boldsymbol{L}' \leftarrow \mathrm{argsort}\,(\boldsymbol{\Theta}_\pm, \textit{ascending}, \textit{row-wise})$ ▷ Sorted angle labels
  $\boldsymbol{L} \leftarrow \boldsymbol{L}'[:, \mathrm{range}\,(n/2)]$ ▷ Only the first $n/2$ columns are needed
  $\boldsymbol{\Delta} \leftarrow \mathrm{sort}\,(\boldsymbol{P}^2, \textit{descending}, \textit{row-wise})$ ▷ Sorted correlation proportions

  $\boldsymbol{l} \leftarrow \emptyset_n$ ▷ Initialize label list of n elements
  for $i \in [1, n]$ **do** ▷ Angle label count for each sample
    $s_\rho \leftarrow 0$ ▷ Initialize cumulative correlation
    $j \leftarrow 0$
    while $s_\rho < \rho$ **do** ▷ Continue until s_ρ satisfies ρ
      $j \leftarrow j + 1$
      $s_\rho \leftarrow s_\rho + \boldsymbol{\Delta}[\,i\,,\,j\,]$ ▷ Add correlation proportion to s_ρ
    end while
    $\boldsymbol{l}[\,i\,] \leftarrow \boldsymbol{L}[\,i\,, \mathrm{range}\,(j)\,]$ ▷ First j component labels for a sample
  end for
  return $\boldsymbol{l}$
end function

It takes data matrix $\boldsymbol{X}$ and hyperparameter *mutual correlation within cluster* ρ as input. Hyperparameter ρ represents the required level of variance each cluster must

explain of its elements. Even though CRUSHES shares the same mathematical background and has a similar cluster structure with CHUNX, the computation of cluster labels for each element is more straightforward in CRUSHES than in CHUNX.

CRUSHES computes the angle labels used for cluster label evaluation similarly as CHUNX does. Thus, lines 8–15 in Algorithm 1 are identical with lines 7–14 in Algorithm 3 – this process is explained in detail in Section 3.3. Beside the sorted angle label matrix $\boldsymbol{L}$, as in Equation (3.12), CRUSHES computes matrix $\boldsymbol{P}^2$, whose rows are the proportions of explained variance of each $\boldsymbol{v} \in \mathrm{col}\,(\boldsymbol{V})$, before starting the actual clustering of $\boldsymbol{X}$. Matrix $\boldsymbol{\Delta}$, whose rows are the rows of $\boldsymbol{P}^2$ sorted in descending order, is computed on line 15 in Algorithm 3.

On lines 18–26 in Algorithm 3, cluster labels are formed for each $\boldsymbol{x} \in \mathrm{row}\,(\boldsymbol{X})$ based on matrices $\boldsymbol{\Theta}_{\pm}$ and $\boldsymbol{\Delta}$. The number of angle labels required for representing a data point $\boldsymbol{x}$ at the level of ρ is determined in a while-loop on lines 21–24 in Algorithm 3 – the proportions of explained variance are iteratively summed up (line 23) starting from the most significant component until the value of ρ is exceeded. Since each eigenvector has either positive or negative coefficient in the linear combination of each $\boldsymbol{x} \in \mathrm{row}\,(\boldsymbol{X})$, the angle labels defining each cluster label can be determined by selecting the number of required columns from the rows of $\boldsymbol{\Theta}_{\pm}$ (line 25 in Algorithm 3).

As the cluster labels are defined for each $\boldsymbol{x}$, CRUSHES returns list $\boldsymbol{l}$ of angle label variations. The unique labels of CRUSHES clusters can be determined by finding the unique variations of $\boldsymbol{l}$ from which a CRUSHES tree structure, illustrated in Figure C.2, can be generated. Since CRUSHES does not produce clusters in a recursive manner as CHUNX does, the formed clusters are not necessarily the leaves of the formed tree structure.

4. EMPIRICAL RESULTS

This chapter evaluates the cluster analyses of CHUNX and CRUSHES for a real-life high-dimensional data set and compares them with k-means, DBSCAN and ORCLUS cluster analyses performed for same data set. For each algorithm the process of selecting the satisfactory hyperparameter configuration, the suitability for particular problem, and the quality of produced clusters are evaluated. The empirical data used in the evaluation is sequential time series data – the data samples with similar shape should belong to the same cluster.

The data used in cluster analysis was preprocessed into a suitable form using `Python` programming language [50] with the help of its packages, *NumPy* [61] and *Pandas* [44], for numerical and statistical computing. The implementations of CHUNX and CRUSHES were programmed using the same technologies. All the figures related to empirical results are visualized using *Matplotlib* [31] – a package for plotting in `Python`.

The clustering results of k-means and DBSCAN are computed with *scikit-learn* project's [45] `Python` implementations for k-means algorithm [54] and DBSCAN algorithm [53]. The results of ORCLUS were computed exceptionally in `R` programming language [48] using its *orclus* package [59], since there were no ORCLUS implementations available in `Python`.

4.1 Empirical Data

The data used in empirical analysis is derived from Finnish energy consumption data. The data consists of energy consumption time series whose values are in the unit of kW h from different types of metering points such as detached houses, terraced houses, blocks of flats, municipal real estates, factories of different manufacturing industries, summerhouses, public lighting, agrarian real estates, business spaces, health care real estates, and link stations. The data has been collected from 6959 metering points between the midnight of the 6^{th} of January 2014 and the midnight of the 23^{rd} of May 2016 consisting of 125 full weeks starting from Monday. The energy consumption values of each time series have been collected on the hour and

their time stamps are in *Coordinated Universal Time* (UTC).

A sample of the energy consumption data is a time series which typically can be decomposed into a macro-scale *seasonal component*, a micro-scale *repeating pattern component*, and a stochastic *noise component*. Seasonal component represents the total consumption at certain time interval $[t_s, t_e]$, where t_s is the start time and t_e the end time of the time interval, and repeating pattern component the relative distribution of consumption over $[t_s, t_e]$. Thus, each energy observation of a sample time series can be represented as function of time t:

$$E(t) = (\mu(t) + \eta(t))E_{\text{seasonal}}(t),$$

where $t_s \leq t \leq t_e$, $E_{\text{seasonal}}(t)$ denotes the total energy consumption at time interval $[t_s, t_e]$, $\mu(t)$ the mean proportion of consumed energy at t from the total energy of the interval, and $\eta(t)$ the noise component. Figure 4.1 illustrates a decomposition of a sample energy curve $E(t)$ into its seasonal component $E_{\text{seasonal}}(t)$ and repeating pattern component $\mu(t) + \eta(t)$. The seasonal component $E_{\text{seasonal}}(t)$, plotted in blue

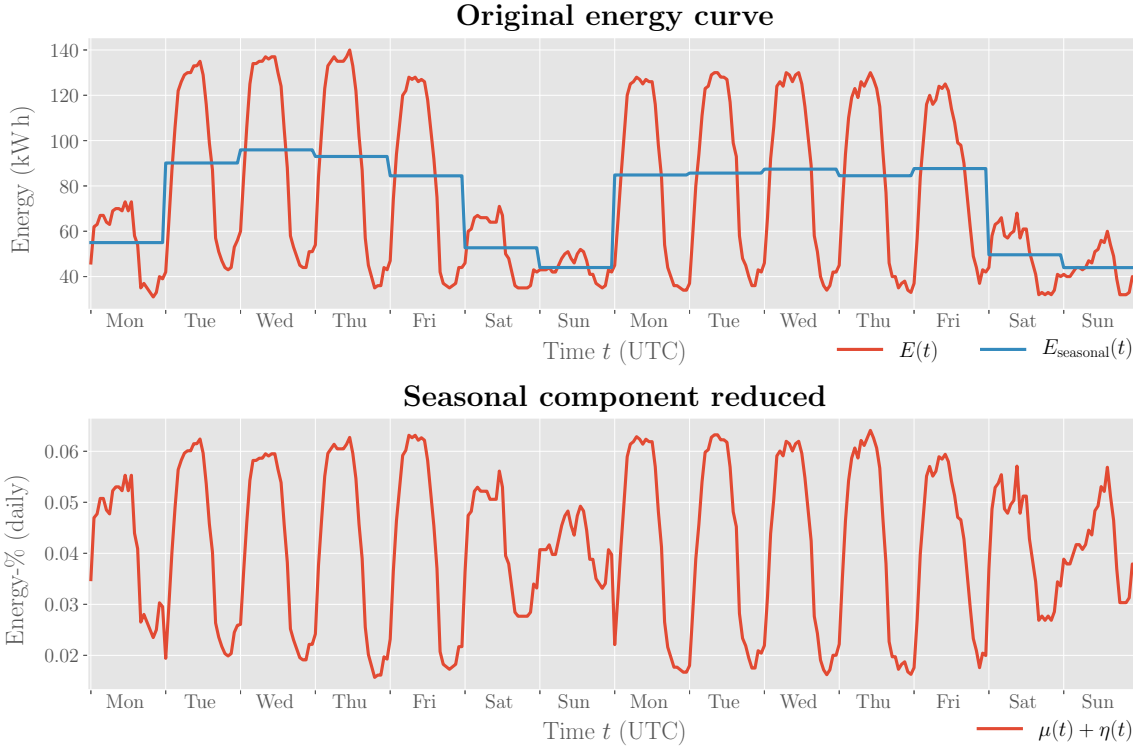

Figure 4.1 *Reducing the seasonal component $E_{seasonal}(t)$ from a sample time series $E(t)$.*

in the upper graph of Figure 4.1, is presented as a time series in which the total energy consumption of each specific day is distributed uniformly over the hours of that day. As the energy values of each day in the sample energy curve are divided by

the corresponding total daily consumption, the resulting curve in the lower graph of Figure 4.1 consists of multiple consecutive relative time distributions of daily energy consumption.

If there is a repeating weekly pattern in the sample energy curve, distributions representing the same weekday should have similar shapes. By computing the average shape of each weekday's time distribution over the complete time range of the data set, the amount of processed data reduces significantly. In order to detect distinguishable shapes of assumed clusters of energy curve behavior, the random variations in the distributions of individual weekdays must be ignored. Figure 4.2 represents the decomposition of consecutive daily time distributions $\mu(t) + \eta(t)$ into repeating weekly pattern in the daily time distribution $\mu(t)$ and noise component $\eta(t)$. Although there seems to be relationship between the variances of $\mu(t)$ and $\eta(t)$

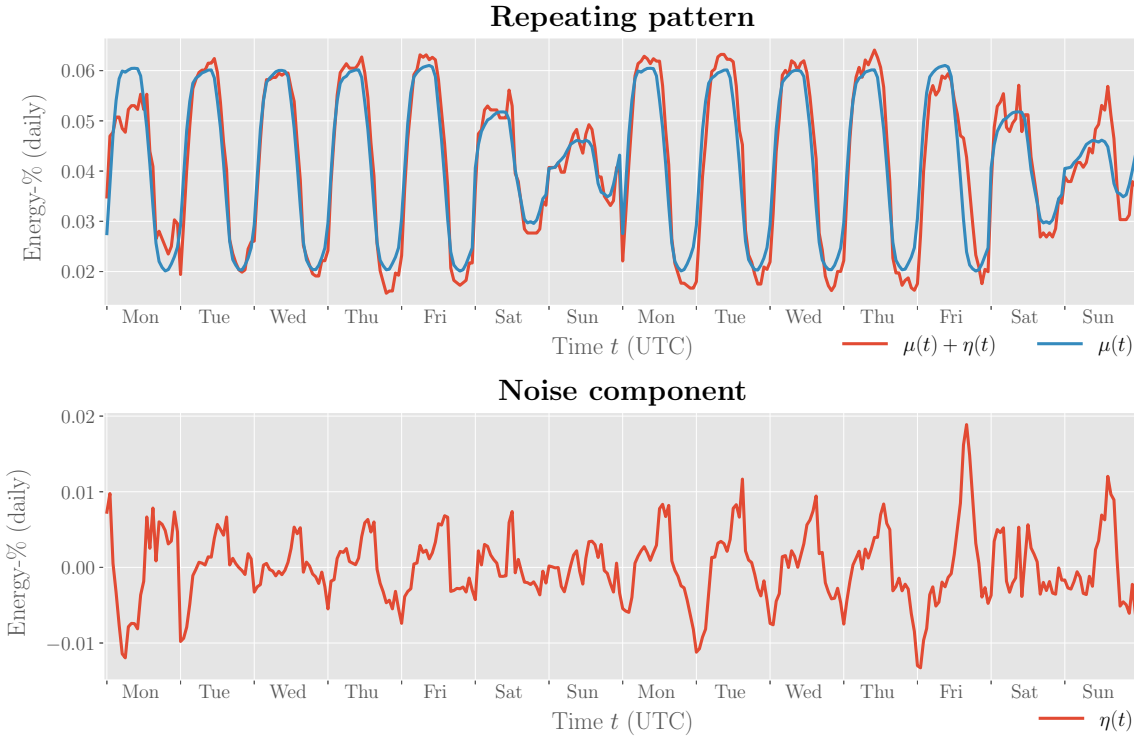

Figure 4.2 *Repeating weekly pattern $\mu(t)$ and noise component $\eta(t)$ of a sample time series.*

in the two-week time range of Figure 4.2 taken from the sample energy curve's data, the relationship is not significant in terms of detecting characteristic shapes.

The repeating pattern of an energy curve can be expressed as a partially defined

function

$$\mu(t) = \begin{cases} \mu_{\text{Mon}}(t), & \text{if } t \in \text{Monday} \\ \mu_{\text{Tue}}(t), & \text{if } t \in \text{Tuesday} \\ \mu_{\text{Wed}}(t), & \text{if } t \in \text{Wednesday} \\ \mu_{\text{Thu}}(t), & \text{if } t \in \text{Thursday} \\ \mu_{\text{Fri}}(t), & \text{if } t \in \text{Friday} \\ \mu_{\text{Sat}}(t), & \text{if } t \in \text{Saturday} \\ \mu_{\text{Sun}}(t), & \text{if } t \in \text{Sunday} \end{cases}, \tag{4.1}$$

where t is in UTC and each weekday is a range of its hours $[0, 23] \in \mathbb{N}$. The relative time distribution of each weekday is illustrated in Figure 4.3. By computing the

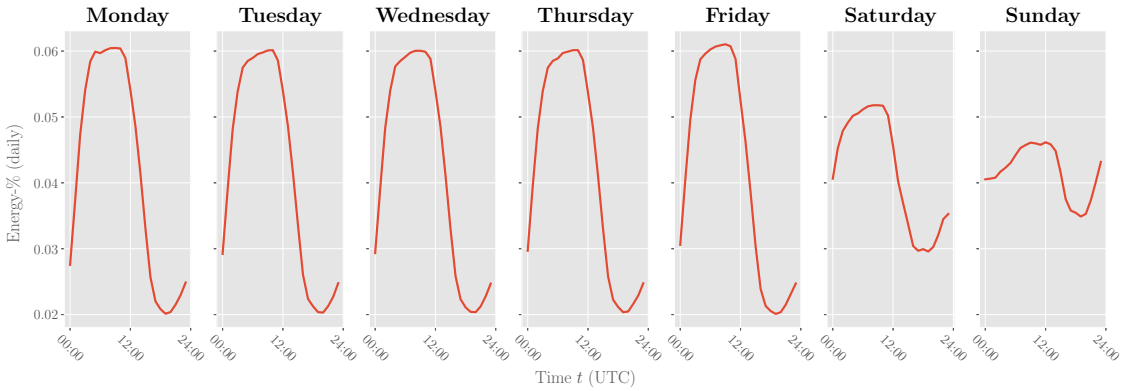

Figure 4.3 *The repeating shapes of each weekdays energy consumption from a sample energy curve.*

value of $\mu(t)$ for each element of $\boldsymbol{t} = \begin{bmatrix} t_1 & \cdots & t_{168} \end{bmatrix}^T$, where t_1 is the first hour of Monday, t_{168} the last hour of Sunday and all the other elements are the ordered hours between t_1 and t_{168}, and dividing the result with the sum of each daily curve, a data sample used in clustering

$$\boldsymbol{x} = \frac{1}{7} \begin{bmatrix} \mu_{\text{Mon}}(t_1) \\ \vdots \\ \mu_{\text{Thu}}(t_{84}) \\ \vdots \\ \mu_{\text{Sun}}(t_{168}) \end{bmatrix} \in \mathbb{R}^{168}. \tag{4.2}$$

Figure 4.4 illustrates an example of $\boldsymbol{x}$ of Equation (4.2) – the shape of the repeating pattern in a energy consumption curve. The complete data set of 6959 energy

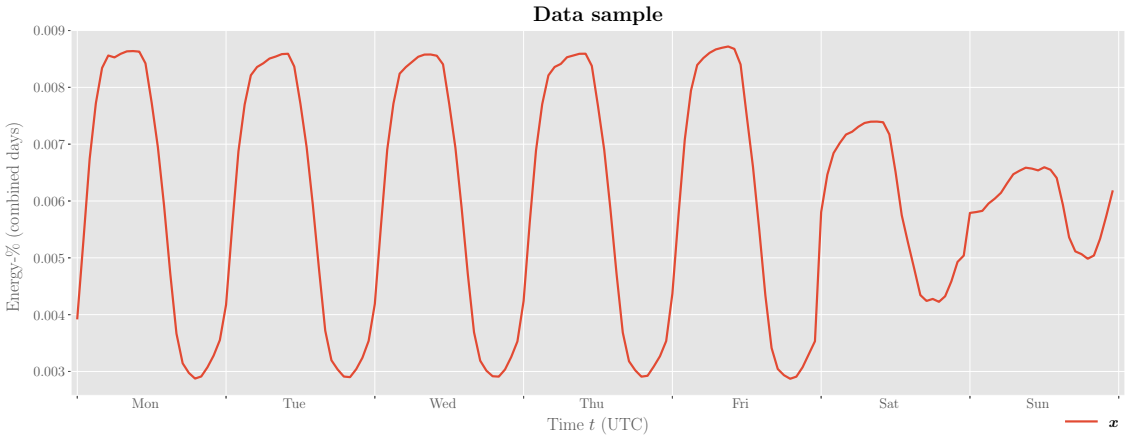

Figure 4.4 *An example of a characteristic shape of the repeating pattern in a energy consumption curve.*

consumption curves is represented as

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{6959} \end{bmatrix}^T \in \mathbb{R}^{6959 \times 168}, \tag{4.3}$$

where each $\boldsymbol{x}_i \in \mathbb{R}^{168}$ is in the format of Equation (4.2) and $i \in \{1, \ldots, 6959\}$. Thus, $\boldsymbol{X}$ corresponds the matrix of Equation (2.2). Figure 4.5 illustrates $\boldsymbol{X}$ as a surface formed by the shapes of repeating patterns in the energy curves. Even though the surface of Figure 4.5 is noisy, it is obvious that a large number samples in $\boldsymbol{X}$ share a common overall shape: There is more consumption in the daytime than at night.

When clustering the shapes of energy consumption curves, it is crucial to use only their current time alignment. From the perspective of energy production and marketing, it is important to know the timing of energy demand in the grid [49]. If the shapes of repeating patterns in energy curves cluster well, the resulting information could be beneficial for forecasting dynamically overall energy demand without complete hour-based energy consumption data.

4.2 Results with k-means

As mentioned in Section 2.2 on page 8, k-means is the most well-known clustering algorithm mainly due to its intuitive user parameter setting and interpretable results – the number of resulting clusters is strictly the predefined k found at a local optimum of E given in Equation (2.7). If a metric is used as a measure of distance, the resulting clusters are represented directly by their centroids. On the other hand, the determination of the right or even suitable k can be a challenging and computationally exhaustive task. Noise in the input data will also affect the quality of

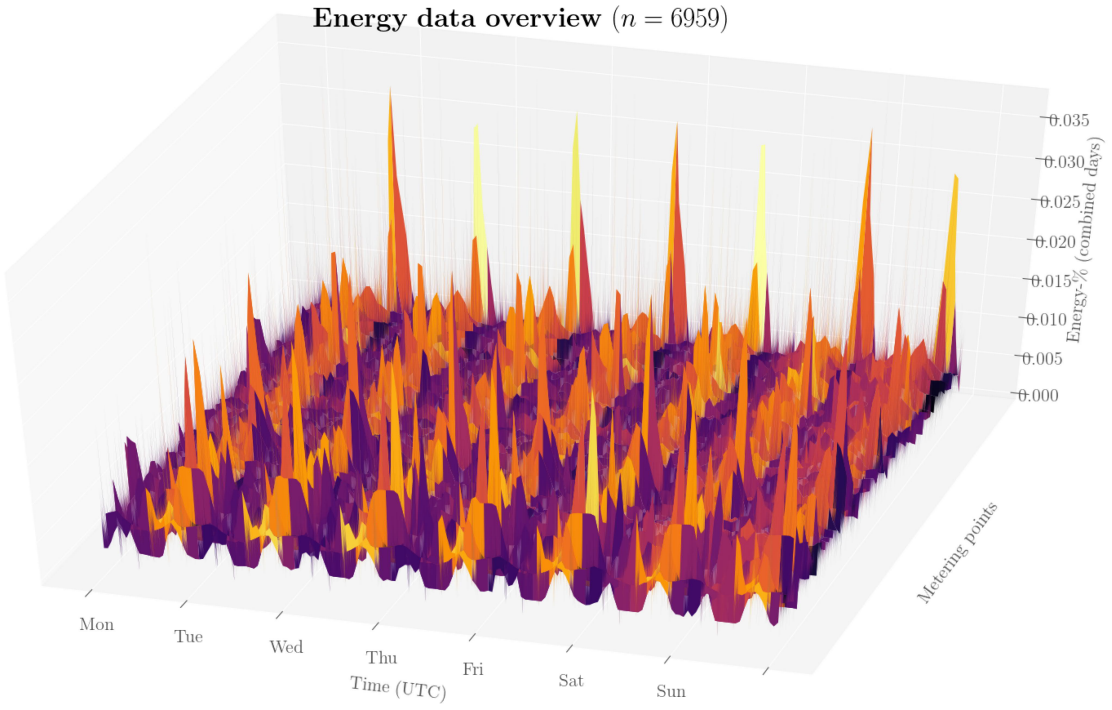

Figure 4.5 *Relative energy consumption data over hours of week from* 6959 *metering points.*

resulting clusters as k-means does not have any mechanisms for detecting outliers. If there is no information available about the internal relationships of data points, it can be hard to decide whether a cluster is a collection of similar data points or noise.

Since k-means takes its input data as a data matrix of Equation (2.2), the energy consumption data of Equation (4.3) is passed to k-means as such. The implementation of k-means used in this evaluation [54], uses the Euclidean distance of Equation (2.3) by default to compute the distances between data points. All the other parameters of the k-means implementation [54] except the desired number of clusters k take their default values in the computations – these parameters are used for fine-tuning the computation and convergence of k-means.

In situations, where there is no prior information about the number of groups in the input data, the desired number of resulting clusters k has to be inferred from the clustering results run with different values of k. A common way of determining the suitable k for k-means is to find a value of k for input data where the value of Equation (2.7) does not improve significantly as k is incremented [33]. Figure 4.6

illustrates the relationship of E of Equation (2.7) and k. Each value of E for different

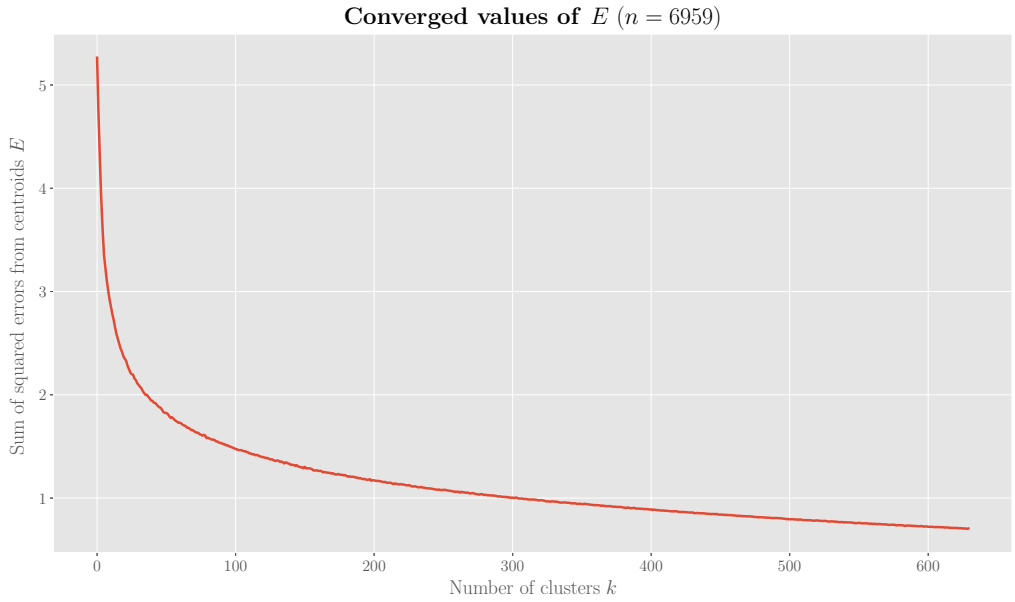

Figure 4.6 *The converged values of E for different values of k.*

k is a mean value of 5 consecutive runs with the same value of k. As it can be seen from Figure 4.6, there is no clear point – an "elbow" – from which the value k could be determined by finding a local minimum from the rate of change of E. For the computations of the empirical evaluation, let $k = 50$.

The first 10 clusters of k-means for energy consumption shape data of Equation 4.3 are illustrated as separate surfaces of different colors in Figure 4.7.[1] The remaining 40 clusters are illustrated in the figures of Appendix A. The resulting clusters of k-means for energy consumption data are surprisingly good when the suitable number of clusters has been determined. The most of the produced clusters represent distinct shapes of repeating patterns of energy consumption. There seems to be clusters of data points without a clear repeating patterns.

Even though the k-means clusters are intuitively recognizable, k-means does not enlighten the relationships between the resulting clusters. Without any noise detection mechanisms, it can be quite unclear whether some clusters represent a distinct subgroup of data or are they just collections of outliers. In addition, k-means as an

[1]The surface plots visualizing clusters have various view angles due to invalid rendering in *mplot3d* library of Matplotlib [43].

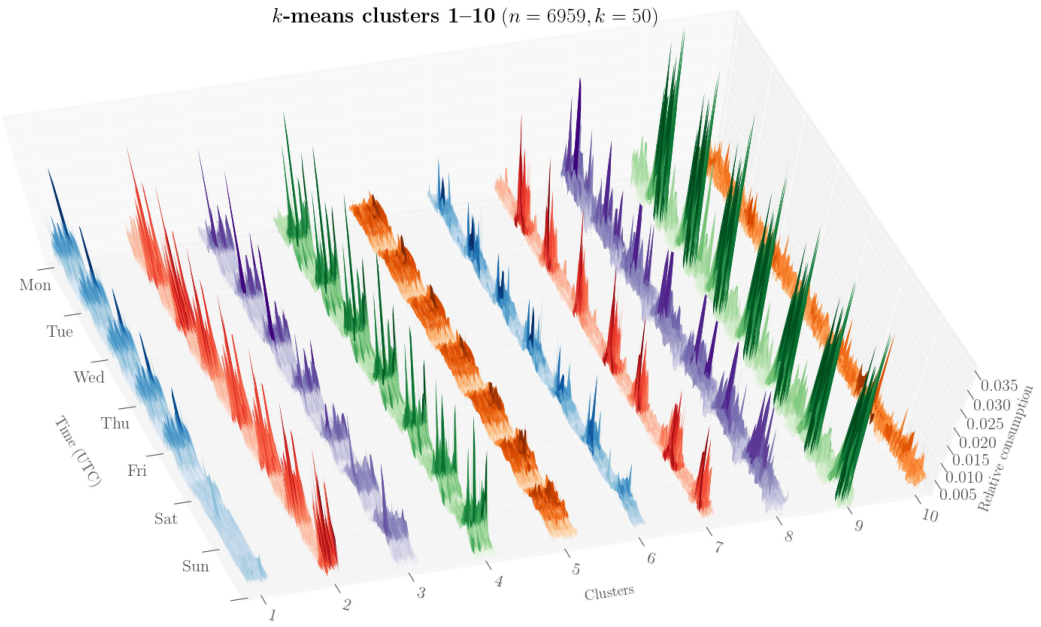

Figure 4.7 *The shapes of the first* 10 *k-means clusters.*

iterative algorithm which finds a local optimum, produces results which can significantly differ from each other between the runs with the same value of k. Despite its intuitive user interface and convenient empirical results, k-means is not suitable for automated, reliable energy consumption pattern profiling due to its exhaustive process of selecting satisfactory k and lack of structures explaining the relationships between clusters.

4.3 Results with DBSCAN

As described in Section 2.2 on page 9, DBSCAN forms clusters from the dense regions of input data. Unlike in k-means, where the clusters have spherical shape, DBSCAN clusters have arbitrary shapes. Since DBSCAN reduces noise from the data, each cluster has to satisfy criteria set by $MinPts$ and ε. Without the prior setting of expected k clusters, DBSCAN has potential of detecting high quality clusters of different sizes.

Since implementation of DBSCAN used in the evaluation [53] supports the use precomputed distance data as input data, the angle distance matrix D_{angles} given in Equation (2.5) is computed from the input data given in Equation (4.3). Before computing D_{angles}, $\boldsymbol{X}$ is centered and normalized as in Equation (3.2).

Similarly as in k-means, the determination of a suitable parameter setup can be a challenging task if there is no prior information about the input data. A method for determining the satisfactory ε for certain $MinPts$ without computing clustering results of DBSCAN beforehand has been proposed by Elbatta et al. [20, p. 127]. In this method, for a certain value of $k = MinPts$, the distance of each data point of input data $\boldsymbol{X}$ to its k^{th} nearest neighbor is computed. As these distances are sorted in ascending order, there should be a similar "elbow" point in the curve as explained in Section 4.2 – the value of suitable ε for chosen $MinPts$. [20, p. 127]. The density of produced clusters decreases rapidly if the value of ε is chosen beyond the "elbow" point. Figure 4.8 illustrates the selection of ε and $MinPts$ as a point on a surface which represents the sorted distances to the k^{th} nearest neighbor for each value of $MinPts$. As can be seen in Figure 4.8, the differences between angles between

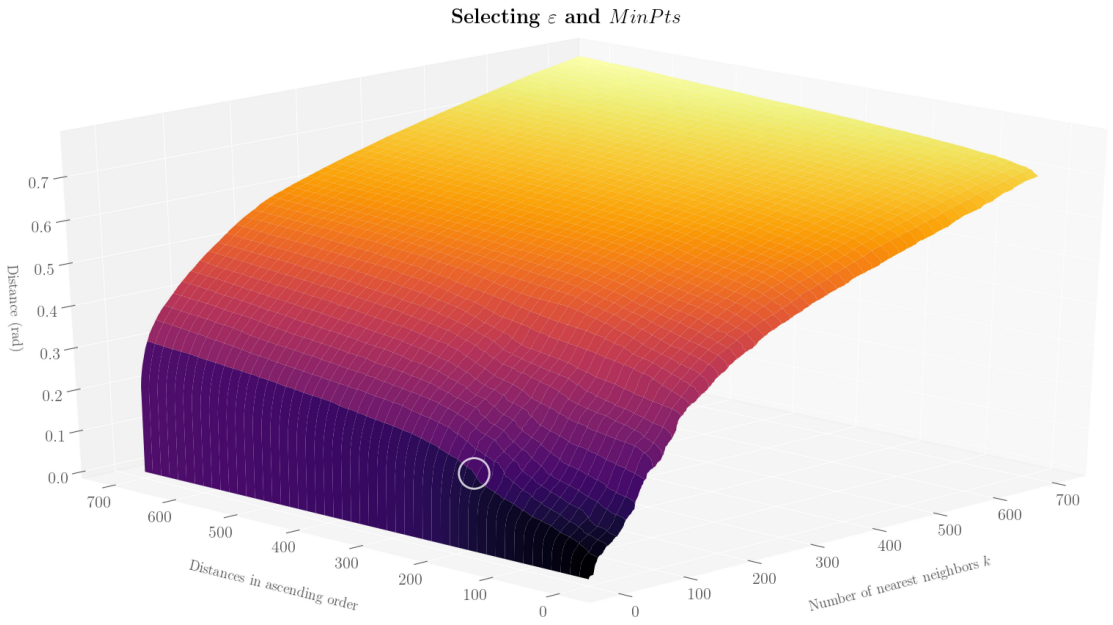

Figure 4.8 *The distances to the k^{th} nearest neighbor for all values of k represented as a surface. The white circle on the surface represents the chosen values of ε and $MinPts$ for clustering the empirical data.*

data points are subtle and there are no clear "elbow" points on the surface. In the chosen parameter setup, $\varepsilon = 0.18$ and $MinPts = 8$. The white circle in Figure 4.8 represents the point which determines the parameter values. The values beyond the limits of axes in Figure 4.8 are not visualized since they increase smoothly through the whole surface.

With the chosen parameter setting of DBSCAN, only 6 clusters were produced. The

clusters of DBSCAN for energy consumption shape data of Equation 4.3 are illustrated as separate surfaces of different colors in Figure 4.9. The clusters produced

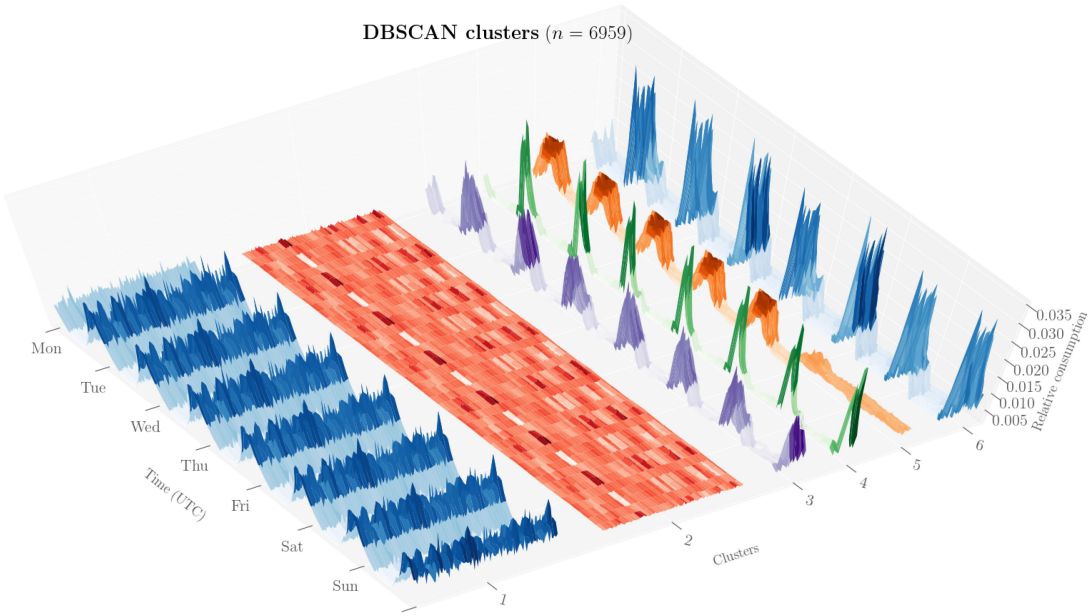

Figure 4.9 *The shapes of the DBSCAN clusters.*

by DBSCAN are definitely distinct shapes but the there are only 6 clusters and 6653 data points marked as noise leaving only the total of 306 points in clusters. The previous results of k-means reveal that there are certain shapes, represented by multiple data points, DBSCAN considers as noise.

Even though DBSCAN detects clusters without prior number of clusters given with selected ε and $MinPts$, it seems to be extremely sensitive to Problem 2. This results into clusters which have only a few members and the most of the data is marked as noise. To improve the results produced by DBSCAN, it would be necessary to select another method for determining suitable ε and $MinPts$. In addition, without any information about the hierarchical relationships of data points, DBSCAN is not suitable to be used on its own for clustering high-dimensional shapes.

4.4 Results with ORCLUS

Contrary to k-means and DBSCAN, ORCLUS, as a correlation clustering algorithm, is designed for detecting clusters in high-dimensional spaces. ORCLUS requires at least three parameters k, k_0, and l to be set. Fortunately, authors of ORCLUS have

proposed a method for determining suitable value of projected dimensionality l [6]. To measure the quality of clusters at certain projected dimensionality l, a quality measure called *cluster sparsity coefficient* is computed for a final clustering result:

$$S(C_1, \ldots, C_k, \mathcal{E}_1, \ldots, \mathcal{E}_k) = \frac{1}{k} \sum_{i=1}^{k} \frac{R(C_i, \mathcal{E}_i)}{R(U, \mathcal{E}_i)} \tag{4.4}$$

where R is the projected energy of the produced clusters given in Equation (2.14) and U is the complete set of data points. A small value of S indicates a good selection of l. Aggarwal and Yu [6] suggest defining a certain threshold for S and selecting the value of l which produces a value of S which is just below the threshold.

In order to run ORCLUS, the empirical data $\boldsymbol{X}$ given in Equation (4.3) does not need normalization. Unfortunately, a bug was found from the implementation of orclus package [59] while running the cluster analyses with different parameter settings: If k-means, assigning the initial number k_0 of eigensystems, ends up with a result with single-element clusters, the covariance matrix cannot be computed for every cluster and the algorithm will terminate unexpectedly. Due to this unpredictable behavior of orclus, comprehensive algorithm parameter setting evaluation could not be performed. The ORCLUS cluster analysis was done using $k = 30$, $k_0 = 35$, and $l = 160$. The value of k was meant to be chosen similarly as in Section 4.2, but due to the bug in the implementation, the largest values the implementation did not crash were $k = 30$ and $k_0 = 35$. The value of l was chosen as the smallest value ORCLUS converged with and did not crash during computation.

The first 10 clusters of ORCLUS for energy consumption shape data of Equation 4.3 are illustrated as separate surfaces of different colors in Figure 4.10. The remaining 20 clusters are illustrated in the figures of Appendix 5. For the resulting ORCLUS clusters, $S(C_1, \ldots, C_{30}, \mathcal{E}_1, \ldots, \mathcal{E}_{30}) = 0.262614$. ORCLUS certainly detects distinct shapes from empirical data $\boldsymbol{X}$ and represents them as consistent clusters. As a clustering algorithm which requires a prior selection for the resulting number of clusters, ORCLUS does not clearly distinguish outlier data points. Similarly as in k-means, ORCLUS merges the outliers with the clusters they are the most similar with and seems to produce quality clusters.

Even though a complete parameter selection process could not be performed for ORCLUS, it is clear that ORCLUS is not suitable for clustering high-dimensional data in applications where the parameter setup needs to be automated: The selection of k and k_0 combined with the computationally exhaustive runs of ORCLUS for determining a suitable l require too much computational resources and information

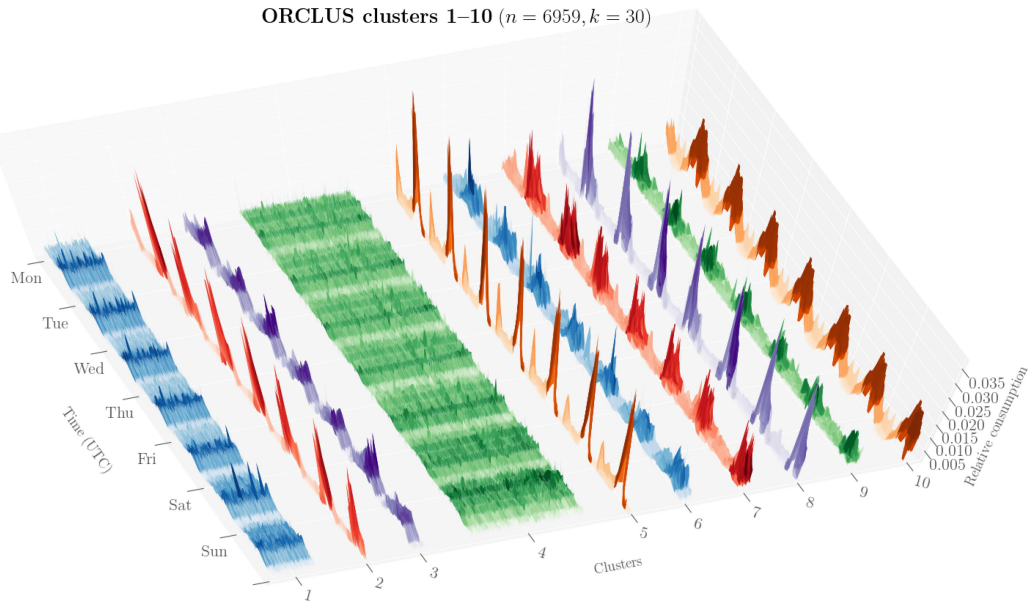

Figure 4.10 *The shapes of the first* 10 *ORCLUS clusters.*

about the data to be clustered. Since ORCLUS does not produce any information about the hierarchical relationships between the data points, it is not even benefitial to use ORCLUS with overestimated values k and k_0. ORCLUS seems to useful in applications of data mining where the number of natural clusters are known beforehand.

4.5 Results with CHUNX

As explained in Section 3.3, CHUNX requires the centering and normalization of its input data – data matrix $\boldsymbol{X}$ given in Equation (4.3) must be modified to satisfy Equation (3.2). Since the CHUNX produces a hierarchical structure explaining the relationships between the data points, its only required input hyperparameter, relative maximum cluster size $p_{\max}$, works as a limit which terminates the hierarchical structure formation. The optional hyperparameter, maximum depth of recursion $d_{\max}$, serves the similar purpose of setting a limit for the depth of recursion in CHUNX-PARTITION function which corresponds to the depth of of the produced hierarchical tree structure. In other words, it is only important to find the satistactory level of clustering instead of finding a single suitable parameter setting as in k-means, DBSCAN and ORCLUS. If the produced CHUNX clustering is too detailed, the clusters with the same parent nodes could be merged as bigger clusters.

In addition, the formed tree structure offers an elegant way of detecting outliers: Instead of deciding whether a data point is an outlier during the run of the algorithm, the produced tree structure can be pruned afterwards by removing clusters under certain size or with certain components labels.

The clustering results of CHUNX were produced with $p_{\max} = 0.1$ without setting the optional $d_{\max}$ and there were 213 clusters produced. The first 10 CHUNX clusters of $\boldsymbol{X}$ whose size n_{C_i}, where $i = 1, \ldots, 58$, is greater than 10 are illustrated as separate surfaces of different colors in Figure 4.11. The remaining 48 clusters clusters are

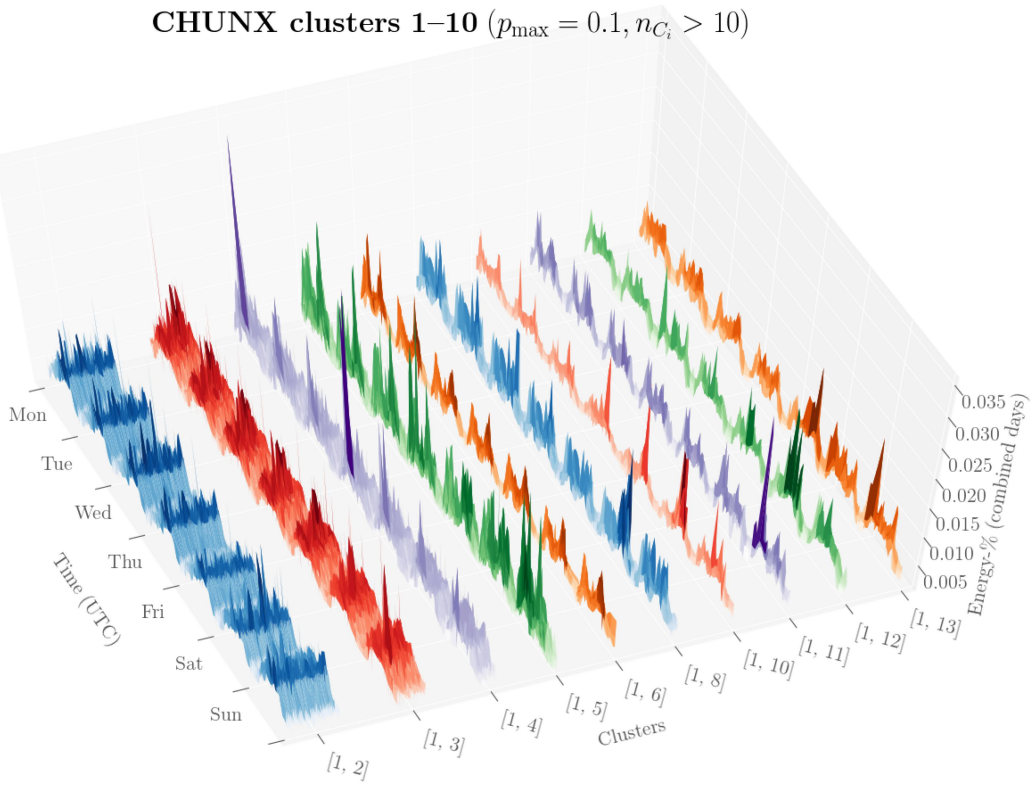

Figure 4.11 *The shapes of the first* 10 *CHUNX clusters which have a size greater than* 10.

illustrated in the figures of Appendix 5. Figure C.1 of Appendix 5 visualizes the hierarchy between the produced clusters as a tree. Each node of the tree, except the root node, represents a distinct signed component in the orthogonal basis of input data $\boldsymbol{X}$. The component labels are in a descending order by their significance in the data set. The data point counts of each cluster are presented below the last component label of each branch of the tree in Figure C.1 – the nodes without a cluster count do not represent a cluster. In Figure C.1, only the clusters which have

$n_{C_i} > 10$ are shown. Besides generating a tree structure, the cluster labels can be used for sorting the clusters by their similarity. Figure 4.12 illustrates the CHUNX clusters sorted by their cluster labels. The surface of Figure 4.12 contains all data

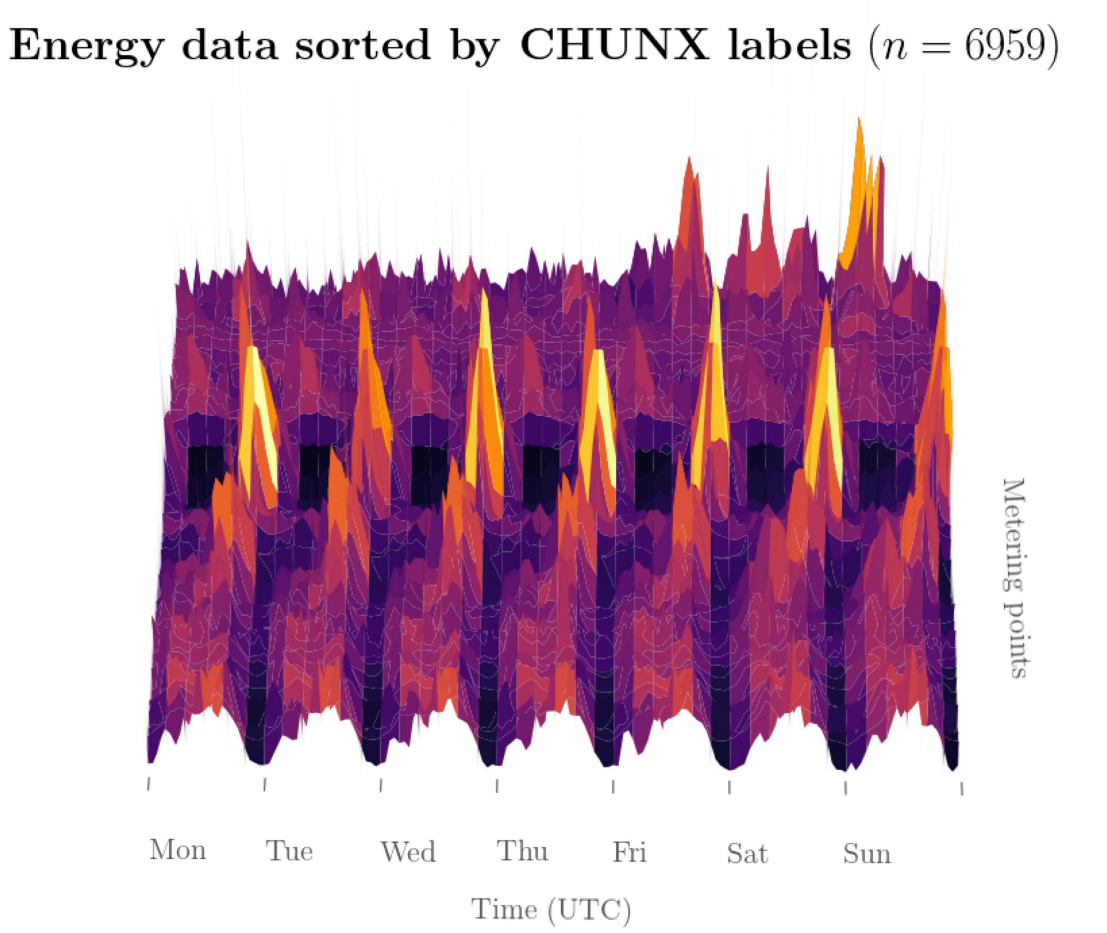

Figure 4.12 *The energy data sorted by CHUNX cluster labels.*

points of $\boldsymbol{X}$.

Since each cluster in CHUNX result is a label variation of components from the orthogonal basis of $\boldsymbol{X}$, each data point can be represented by the shapes each component label represents. Figure 4.13 illustrates the components representing a cluster $[-2, +1]$ and Figure 4.14 the reconstruction of a data point using only the components of data point's cluster $[-2, +1]$. The cluster components can be used for creating a shape representing the whole cluster by computing the mean proportions of each component from the individual component proportions each data point.

CHUNX is turns out to be a useful algorithm for detecting high-dimensional shapes and fundamental relationships relationships between data points and its easy-to-

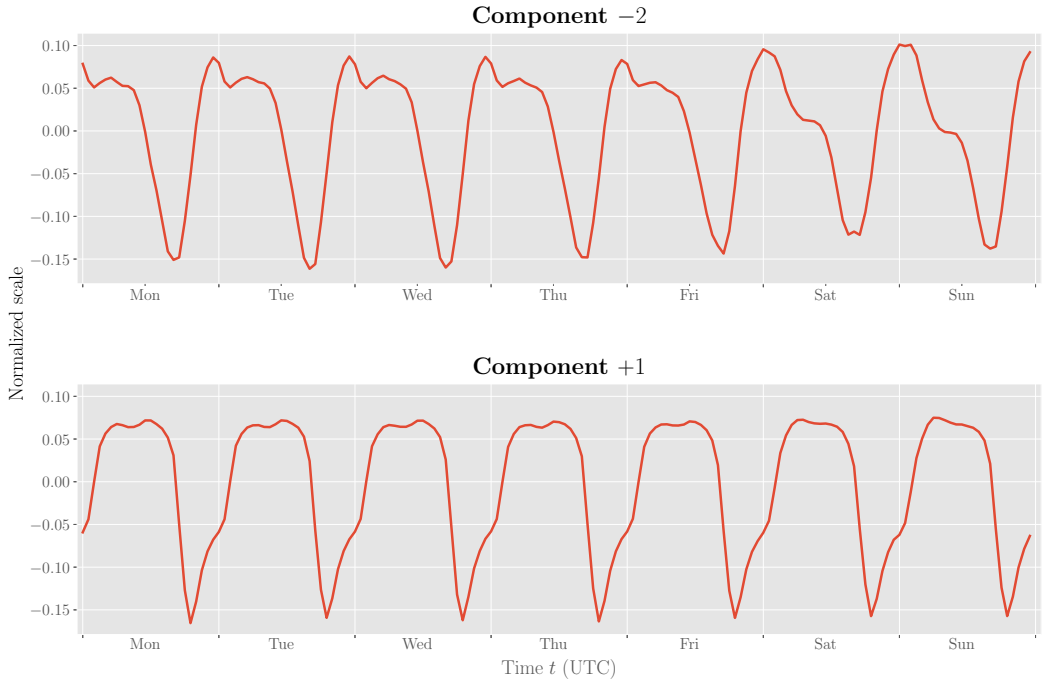

Figure 4.13 *Components* -2 *and* $+1$ *of data set* $\boldsymbol{X}$.

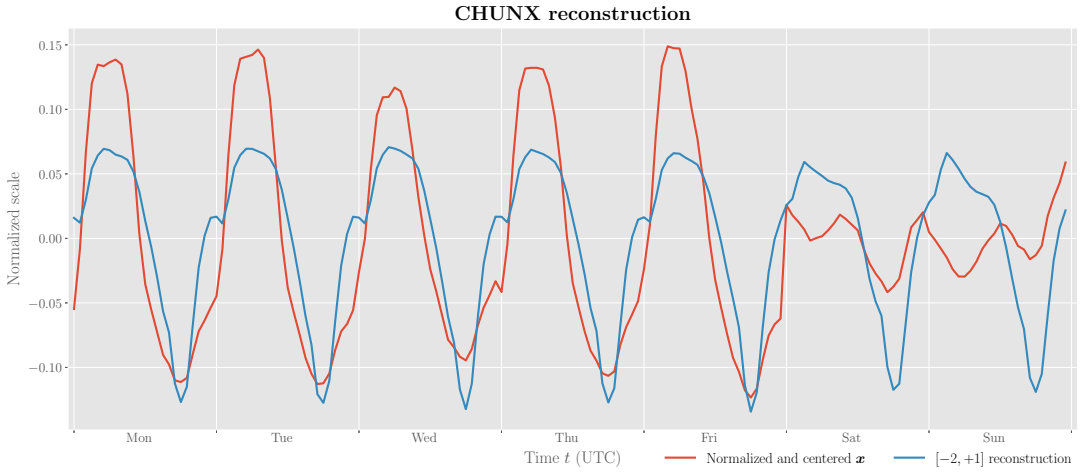

Figure 4.14 *Data point reconstruction using only the components of cluster* $[-2, +1]$ *in CHUNX.*

use and flexible parameter setup is suitable for applications in which the parameter setup needs to be automated. In addition, the resulting hierarchical clusters offer a broad range of possibilities to detect noise and to merge small clusters into larger ones.

4.6 Results with CRUSHES

Even though the implementation and functionality of CHUNX is similar to CRUSHES, their main difference lies in the semantics of their hyperparameters – CHUNX clusters data into subgroups under the size set by $p_{\max}$, whereas CRUSHES forms clusters whose data points share the same components explaining a selected proportion variance of a data point ρ. As mentioned in Section 3.1, unlike CHUNX which produces a tree structure in which the clusters are only the leaf nodes, the clusters of a CRUSHES tree are not always the leaf nodes of the tree. Hyperparameter ρ should be set as $p_{\max}$ of CHUNX: It is not necessary to find a single suitable value for ρ but a level which produces results that are detailed enough. Similarly as in the CHUNX clustering, the input data $\boldsymbol{X}$ passed to CRUSHES needs to be centered and normalized.

The clustering results of CRUSHES were produced with $\rho = 0.75$ and there were 3766 clusters produced. The number of produced clusters in CRUSHES may seem large but it should be noted that there are 3448 single-element clusters and 3706 clusters with a size less than or equal to 10. The first 10 CRUSHES clusters whose size n_{C_i}, where $i = 1, \ldots, 58$, is greater than 10 are illustrated as separate surfaces of different colors in Figure 4.15. The remaining x clusters are illustrated in the figures of Appendix 5. The CRUSHES tree structure is illustrated in Figure C.2 of Appendix 5. The tree in Figure C.2 has the same properties as the CHUNX tree of Figure C.1 explained in Section 4.5. Figure 4.16 visualizes the CRUSHES clusters sorted by their cluster labels. The surface of Figure 4.16 contains all data points of $\boldsymbol{X}$. The overviews of sorted CHUNX data in Figure 4.12 and sorted CRUSHES data in Figure 4.16 visualize the similarity in the results of the algorithms.

As the CRUSHES clusters have been formed, each data point can be reconstructed at certain accuracy by using only the components denoted by the cluster label. The data point of Figure 4.14 which belonged to cluster $[-2, +1]$ in CHUNX belongs to cluster $[-2, +1, +4, +3]$ in CRUSHES. Figure 4.17 illustrates the components $+4$ and $+3$ of cluster $[-2, +1, +4, +3]$ and Figure 4.18 illustrates the reconstruction of the data point using only the components of cluster $[-2, +1, +4, +3]$. The components -2 and $+1$ are the components of Figure 4.13. Since CRUSHES forms clusters by the order and proportion of each component, whereas CHUNX uses only the order of a components, the CRUSHES representation is more accurate than the corresponding CHUNX representation.

Even though CRUSHES produces a large number of clusters for the energy consumption shape data set, it turns out to be a powerful tool for exploring detailed

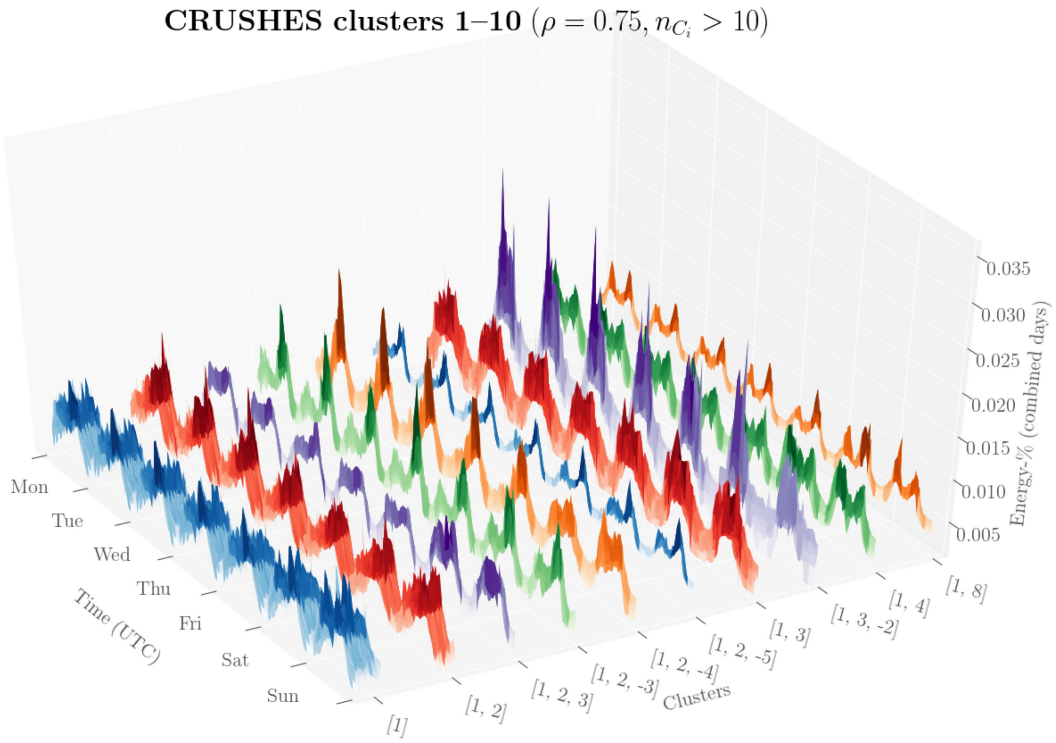

Figure 4.15 *The shapes of the first* 10 *CRUSHES clusters whose size is greater than* 10.

internal relationships of data points. Due to its clustering tree structure in which the clusters can appear in multiple levels, the produced clusters can be filtered even better than in CHUNX. Since hyperparameter ρ ensures that a certain proportion of each data point in a cluster is explained by the components representing the cluster, it is improbable that the produced clusters contain outliers. Straightforward parameter setting, the high accuracy of produced clusters, and clusters which appear on multiple levels of the clustering hierarchy are the key factors which make CRUSHES to stand out from other correlation clustering algorithms as an algorithm which can be implemented as a part of an automated machine learning process.

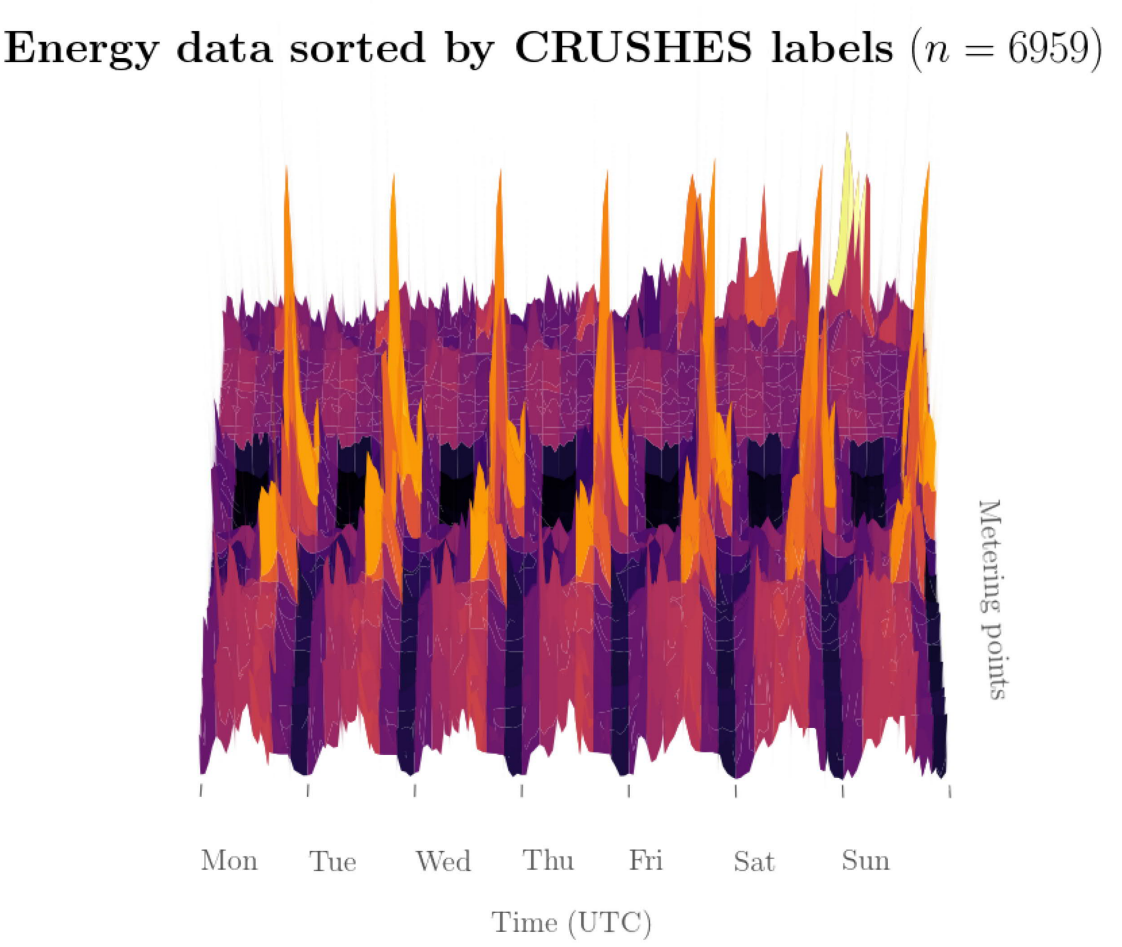

Figure 4.16 *The energy data sorted by CRUSHES cluster labels.*

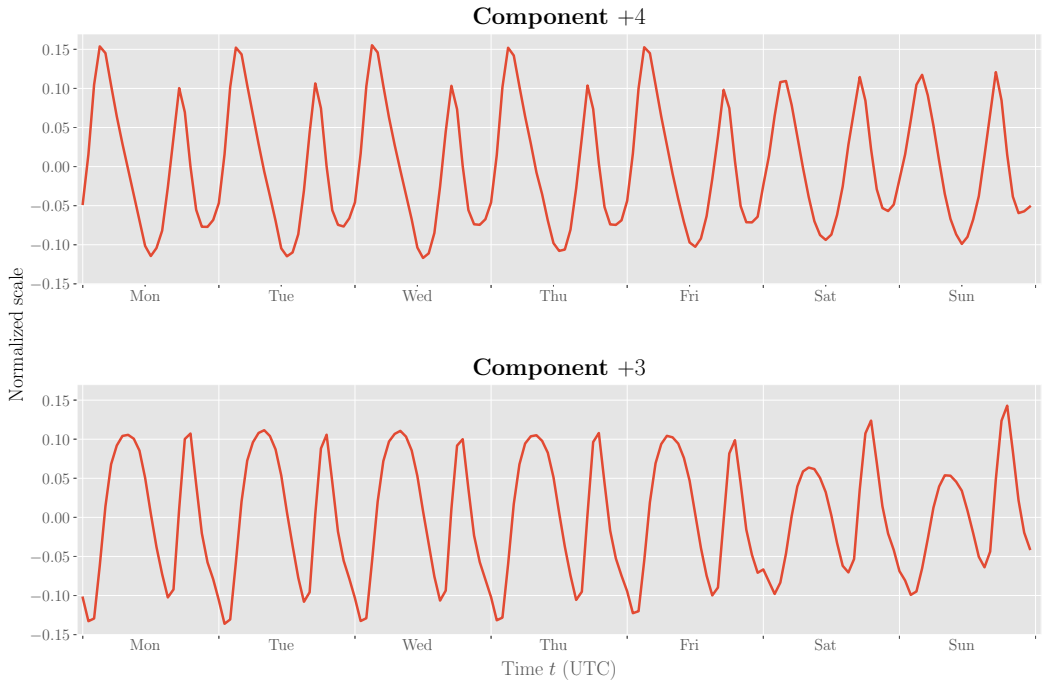

Figure 4.17 *Components* $+4$ *and* $+3$ *of data set* $\boldsymbol{X}$.

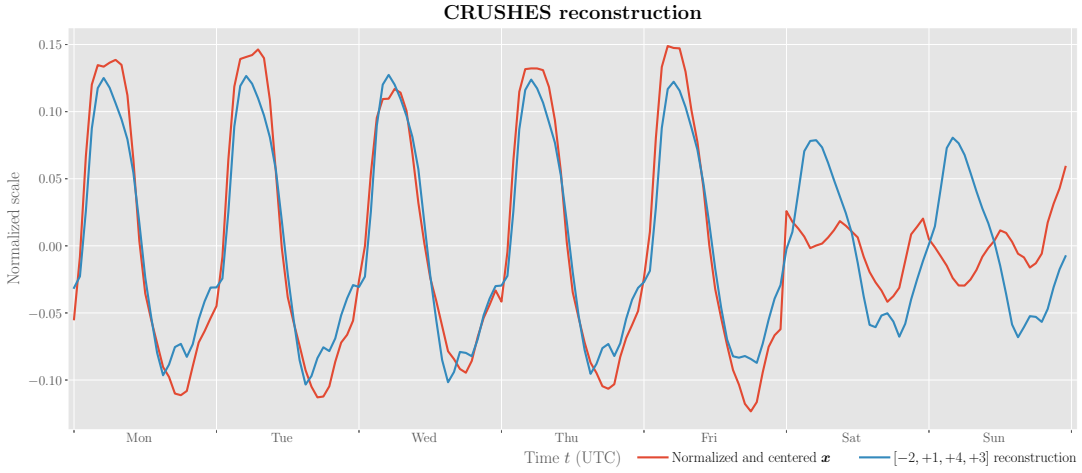

Figure 4.18 *Data point reconstruction using only the components of cluster* $[-2, +1, +4, +3]$ *in CRUSHES.*

5. CONCLUSIONS

Since the amount of produced and collected data has not shown any signs of deceleration in the recent years, it is evident that the applications of advanced data analysis using the techniques of machine learning and AI will become more and more common. Especially, the unsupervised learning methods are likely to play an important role in the predictive analysis in the applications of data sets too large to be manually classified.

Even though the study of unsupervised learning techniques using ANNs has been active in the recent years and has produced astonishing results, it has not been mathematically explained, what actually happens inside the complex ANNs. In addition to the mathematical expressiveness and predictable behavior, clustering algorithms require often less computational resources than ANNs which is why different clustering methods have become common tools of quantitative analysis in many fields of science and engineering.

Since many clustering algorithms are designed for clustering data in at most three dimensions, clustering has not been a popular approach for solving machine learning problems in high-dimensions. The challenges in clustering set by the curse of dimensionality have been acknowledged and ways to overcome the challenges have been proposed. Although the study of data mining has produced multiple powerful algorithms for clustering high-dimensional data, the parameter settings of these algorithms have not served the purpose of using them as a part of an automated application.

In this thesis, two PCA-based correlation clustering algorithms called CHUNX and CRUSHES were introduced. The main objective in the design of CHUNX and CRUSHES was an algorithm which avoids the problems set by the curse of dimensionality in clustering high-dimensional data, provides an intuitive and simple user interface, and produces highly expressive and accurate clustering results. Since both CHUNX and CRUSHES were designed to detect characteristic shapes of curves, they operate only on normalized and centered data. Therefore, CHUNX and CRUSHES are not suitable for applications in which the magnitude of a data point is an im-

portant characteristic feature.

For applications in which the magnitude of data does not play an important role, CHUNX and CRUSHES avoid Problem 2 by operating on angles in arbitrarily oriented subspaces of normalized and centered data, Problem 3 by considering the most significant components of each data point without reducing the components of the data set beforehand, and Problem 4 by not reducing noise during the execution. Although matrix diagonalization done in PCA is a powerful method for detecting relationships between data points, it cannot detect nonlinear dependencies between data points which is why CHUNX and CRUSHES cannot fully overcome Problem 1.

When it comes to the design of an algorithm user interface, both algorithms require only one hyperparameter which defines, how detailed the produced clustering will be. Since the results of both algorithms include a hierarchical structure explaining the relationships between data points, the produced results offer a possibility to improve the clustering results by merging small clusters or removing outliers without rerunning the algorithms.

The empirical evaluation of CHUNX and CRUSHES was conducted by comparing the user experience and clustering results of k-means, DBSCAN, ORCLUS, CHUNX, and CRUSHES. The evaluation ORCLUS could not be fully executed due to a bug in the available algorithm implementation which was discovered during the analysis. The data used in the empirical evaluation consisted of shapes of weekly energy consumption from Finnish energy consumption data. Each algorithm managed to recognize distinct shapes quite well but there were major deficiencies the usability of k-means, DBSCAN, and ORCLUS. Without a hierarchical structure explaining the clustering results, the produced clusters became useless if the chosen parameter setting was not suitable. Contrary to k-means, DBSCAN, and ORCLUS, the hierarchical clusters and flexible parameter settings of CHUNX and CRUSHES offered an easy and expressive way to detect clusters from data and to explore data in which the number of clusters is not known beforehand.

Even without the bug, ORCLUS is not suitable as a part of an automated process. Despite that, further comparisons with CHUNX and CRUSHES need to be executed with functioning algorithms designed for high-dimensional data. There are algorithms such as COPAC [5] and ERiC [4] which seem to outperform ORCLUS in computational efficiency and mathematical expressiveness by providing hierarchical structures of constructed clusters [37, p. 37]. These algorithms are provided only in *ELKI Data Mining* framework which is implemented in `Java` programming lan-

guage [52]. Unfortunately, the framework was discovered too late to be used as a part of this project.

Both CHUNX and CRUSHES propose a new PCA-based way of clustering high-dimensional data without extensive prior knowledge about the input data. The simple user interfaces allow a straightforward usage of both algorithms in applications where the algorithm parameter setup must be automated. The empirical analysis of energy consumption data indicated also, that the hierarchical clustering structures of CHUNX and CRUSHES are extremely useful in filtering outliers off the data. Even though both algorithms stood out in the empirical evaluation, further experiments and more detailed analysis must be performed with other existing algorithms designed for clustering high-dimensional data in order to reveal the true potential of CHUNX and CRUSHES.

BIBLIOGRAPHY

[1] E. Abbena, S. Salamon, and A. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica*. CRC Press, 2006.

[2] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[3] E. Achtert, C. Böhm, J. David, P. Kröger, and A. Zimek, "Robust clustering in arbitrarily oriented subspaces," in *Proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM, 2008, pp. 763–774.

[4] E. Achtert, C. Bohm, H.-P. Kriegel, P. Kroger, and A. Zimek, "On exploring complex relationships of correlation clusters," in *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*. IEEE, 2007, pp. 52–61.

[5] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek, "Robust, complete, and efficient correlation clustering," in *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 2007, pp. 413–418.

[6] C. C. Aggarwal and P. S. Yu, "Finding generalized projected clusters in high dimensional spaces," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 70–81. [Online]. Available: http://doi.acm.org/10.1145/342009.335383

[7] A. A. Alizadeh, M. B. Elsen, R. E. Davis, C. Ma, *et al.*, "Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling," *Nature*, vol. 403, no. 6769, p. 503, 2000.

[8] M. R. Anderberg, *Cluster Analysis for Applications: Probability and Mathematical Statistics: A Series of Monographs and Textbooks*. Academic Press, 2014, vol. 19.

[9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," in *ACM Sigmod Record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.

[10] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 2015.

[11] P. Berkhin *et al.*, "A survey of clustering data mining techniques." *Grouping Multidimensional Data*, vol. 25, p. 71, 2006.

[12] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Proceedings of the International Conference on Database Theory*. Springer, 1999, pp. 217–235.

[13] A. Califano, G. Stolovitzky, Y. Tu, *et al.*, "Analysis of gene expression microarrays for phenotype classification." in *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, vol. 8, 2000, pp. 75–85.

[14] G. A. Carpenter and S. Grossberg, *Pattern Recognition by Self-Organizing Neural Networks*. MIT Press, 1991.

[15] S.-H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *City*, vol. 1, no. 2, p. 1, 2007.

[16] S.-S. Choi, S.-H. Cha, and C. C. Tappert, "A survey of binary similarity and distance measures," *Journal of Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43–48, 2010.

[17] A. Das, H. Singh, and D. Joseph, "A longitudinal study of e-government maturity," *Information & Management*, vol. 54, no. 4, pp. 415–426, 2017.

[18] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, pp. 1–38, 1977.

[19] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 269–274.

[20] M. T. Elbatta and W. M. Ashour, "A dynamic method for discovering density varied clusters," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 6, no. 1, pp. 123–134, 2013.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[22] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*, 5th ed. John Wiley & Sons, Ltd, 2011.

[23] D. Fisher, "Improving inference through conceptual clustering," in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'87. AAAI Press, 1987, pp. 461–465. [Online]. Available: http://dl.acm.org/citation.cfm?id=1856740.1856753

[24] M. Garey, D. Johnson, and H. Witsenhausen, "The complexity of the generalized lloyd-max problem (corresp.)," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 255–256, 1982.

[25] A. Gionis, A. Hinneburg, S. Papadimitriou, and P. Tsaparas, "Dimension induced clustering," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 2005, pp. 51–60.

[26] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd ed. Elsevier, 2006.

[27] R. Haralick and R. Harpaz, "Linear manifold clustering," in *Proceedings of the 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 132–141. [Online]. Available: https://doi.org/10.1007/11510888_14

[28] A. Hinneburg and H.-H. Gabriel, "Denclue 2.0: Fast clustering based on kernel density estimation," in *Proceedings of the 7th International Conference on Intelligent Data Analysis*, ser. IDA'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 70–80. [Online]. Available: http://dl.acm.org/citation.cfm?id=1771622.1771632

[29] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of Educational Psychology*, vol. 24, no. 6, p. 417, 1933.

[30] P. V. Hough, "Method and means for recognizing complex patterns," U.S. Patent 3 069 654, 1962.

[31] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. [Online]. Available: http://aip.scitation.org/doi/abs/10.1109/MCSE.2007.55

[32] D. Kazempour, M. Mauder, P. Kröger, and T. Seidl, "Detecting global hyperparaboloid correlated clusters based on hough transform," in *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. ACM, 2017, p. 31.

[33] D. J. Ketchen Jr and C. L. Shook, "The application of cluster analysis in strategic management research: an analysis and critique," *Strategic Management Journal*, pp. 441–458, 1996.

[34] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[35] E. F. Krause, *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Courier Corporation, 2012.

[36] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "A general framework for increasing the robustness of PCA-based correlation clustering algorithms," in *Scientific and Statistical Database Management*. Springer, 2008, pp. 418–435.

[37] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *The ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 1, pp. 1:1–1:58, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1497577.1497578

[38] S. Lang, "Algebra. number 211 in graduate texts in mathematics," 2002.

[39] K. Lyko, M. Nitzschke, and A.-C. N. Ngomo, "Big data acquisition," in *New Horizons for a Data-Driven Economy*. Springer, 2016, pp. 39–61.

[40] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[41] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: a survey," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 1, no. 1, pp. 24–45, 2004.

[42] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. [Online]. Available: http://nlp.stanford.edu/IR-book/information-retrieval-book.html

[43] "mplot3d FAQ," http://matplotlib.org/mpl_toolkits/mplot3d/faq.html, matplotlib, accessed: 2017-11-16.

[44] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51 – 56.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] D. Pfitzner, R. Leibbrandt, and D. Powers, "Characterization and evaluation of similarity measures for pairs of clusterings," *Knowledge and Information Systems*, vol. 19, no. 3, pp. 361–394, 2009.

[47] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1–16, 2016.

[48] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[49] I. Richardson, M. Thomson, D. Infield, and C. Clifford, "Domestic electricity use: A high-resolution energy demand model," *Energy and Buildings*, vol. 42, no. 10, pp. 1878–1887, 2010.

[50] G. Rossum, "Python tutorial, technical report cs-r9526," *Centrum voor Wiskunde en Informatica, Amsterdam*, 1995.

[51] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[52] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek, "A framework for clustering uncertain data," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1976–1979, 2015. [Online]. Available: http://www.vldb.org/pvldb/vol8/p1976-schubert.pdf

[53] "DBSCAN implementation documentation," http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html, scikit-learn, accessed: 2017-08-24.

[54] "KMeans implementation documentation," http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, scikit-Learn, accessed: 2017-08-24.

[55] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: A multi-resolution clustering approach for very large spatial databases," in *Proceedings of the 24rd International Conference on Very Large Data Bases*, ser. VLDB '98.

San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 428–439. [Online]. Available: http://dl.acm.org/citation.cfm?id=645924.671342

[56] G. W. Stewart, "On the early history of the singular value decomposition," *SIAM Review*, vol. 35, no. 4, pp. 551–566, 1993.

[57] G. Strang, *Linear Algebra and Its Applications.* Belmont, CA: Thomson, Brooks/Cole, 2006.

[58] J. Stutz and P. Cheeseman, "AutoClass – a Bayesian approach to classification," *Fundamental Theories of Physics*, vol. 70, pp. 117–126, 1996.

[59] G. Szepannek, *orclus: ORCLUS subspace clustering*, 2013, r package version 0.2-5. [Online]. Available: https://CRAN.R-project.org/package=orclus

[60] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[61] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011. [Online]. Available: http://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37

[62] W. Wang, J. Yang, and R. R. Muntz, "Sting: A statistical information grid approach to spatial data mining," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 186–195. [Online]. Available: http://dl.acm.org/citation.cfm?id=645923.758369

[63] "DBSCAN-Illustration," https://upload.wikimedia.org/wikipedia/commons/a/af/DBSCAN-Illustration.svg, Wikipedia, accessed: 2017-08-25.

[64] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[65] J. Xu, B. Xu, P. Wang, S. Zheng, G. Tian, and J. Zhao, "Self-taught convolutional neural networks for short text clustering," *Neural Networks*, vol. 88, pp. 22–31, 2017.

[66] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[67] Y. Zhang, H. Wu, and L. Cheng, "Some new deformation formulas about variance and covariance," in *Proceedings of 4th International Conference on Modelling, Identification and Control (ICMIC2012)*. IEEE, 2012, pp. 987–992.

APPENDIX A. K-MEANS CLUSTERS

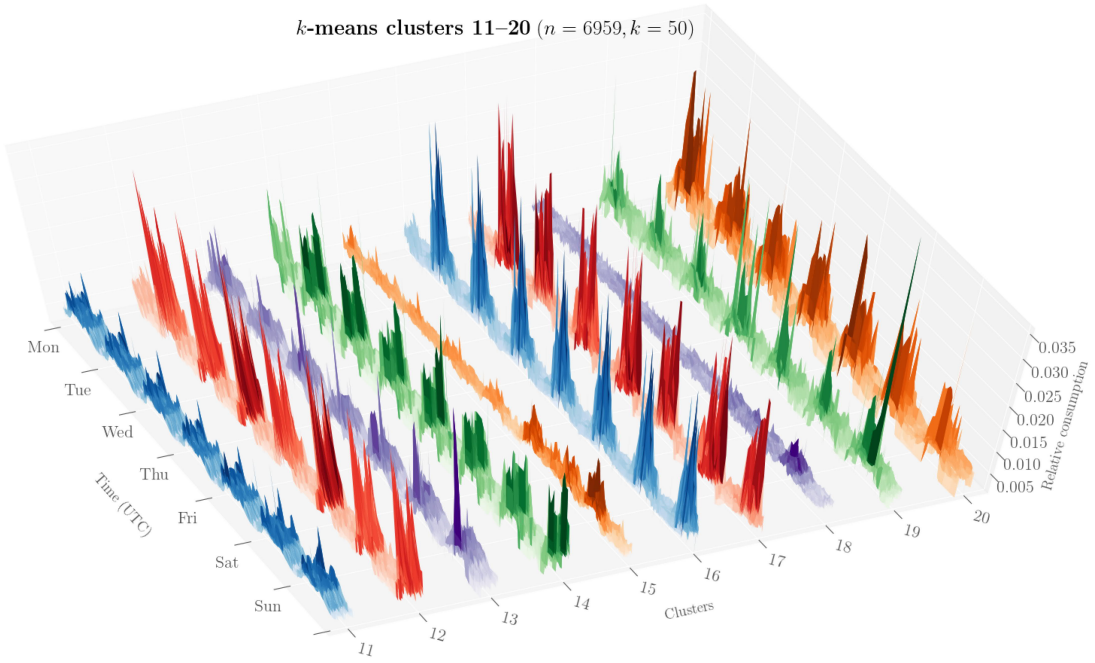

Figure A.1 *The shapes of k-means clusters from* 11 *to* 20.

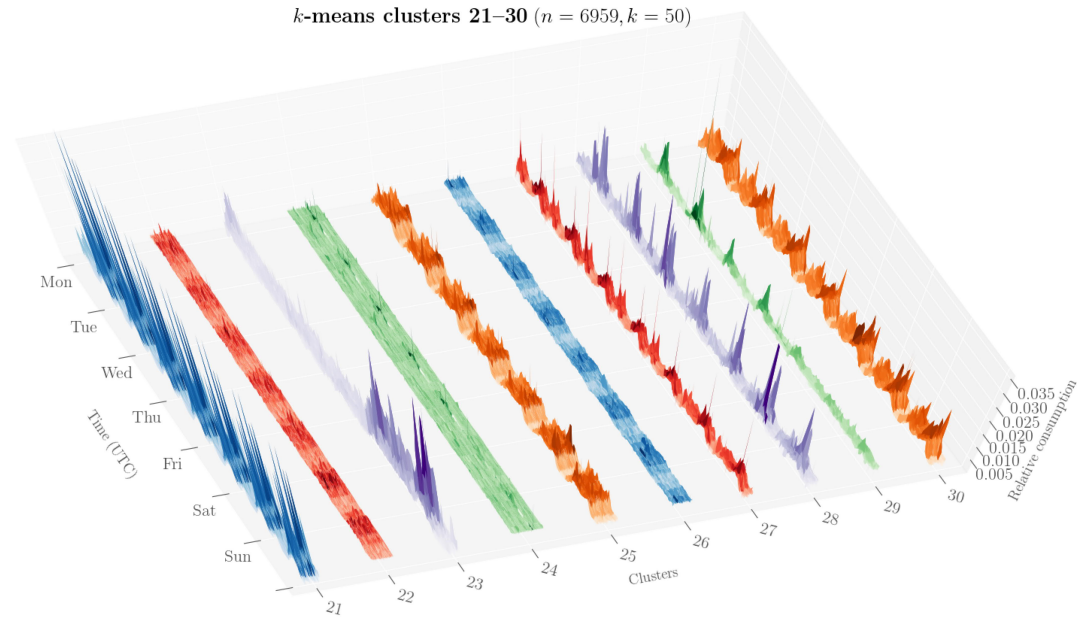

Figure A.2 *The shapes of k-means clusters from* 21 *to* 30.

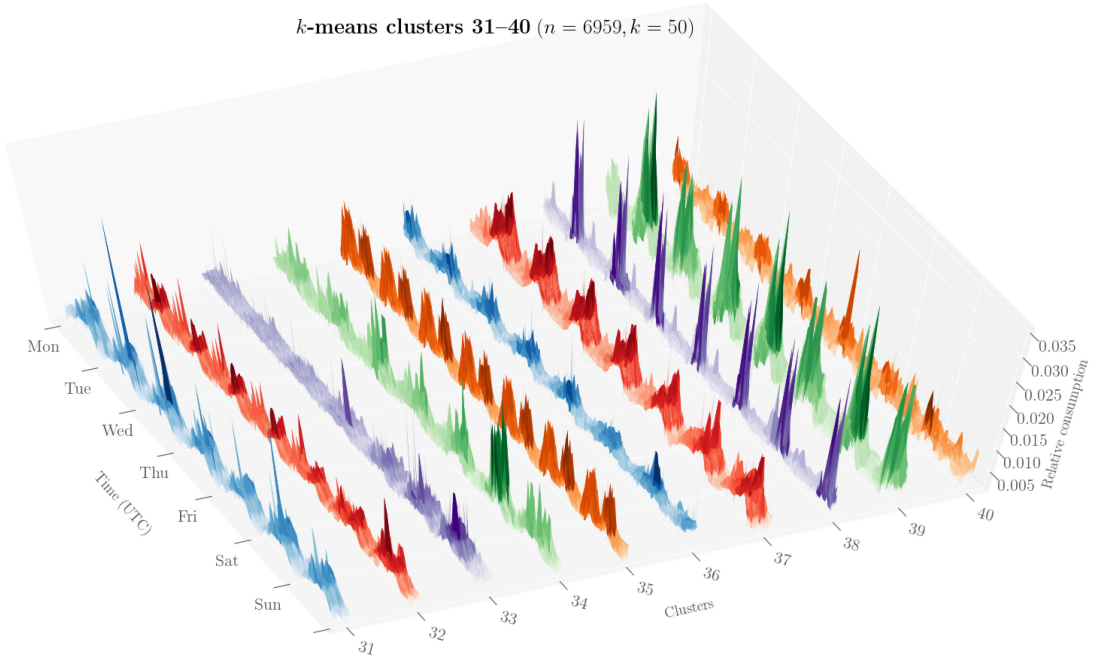

Figure A.3 *The shapes of k-means clusters from* 31 *to* 40.

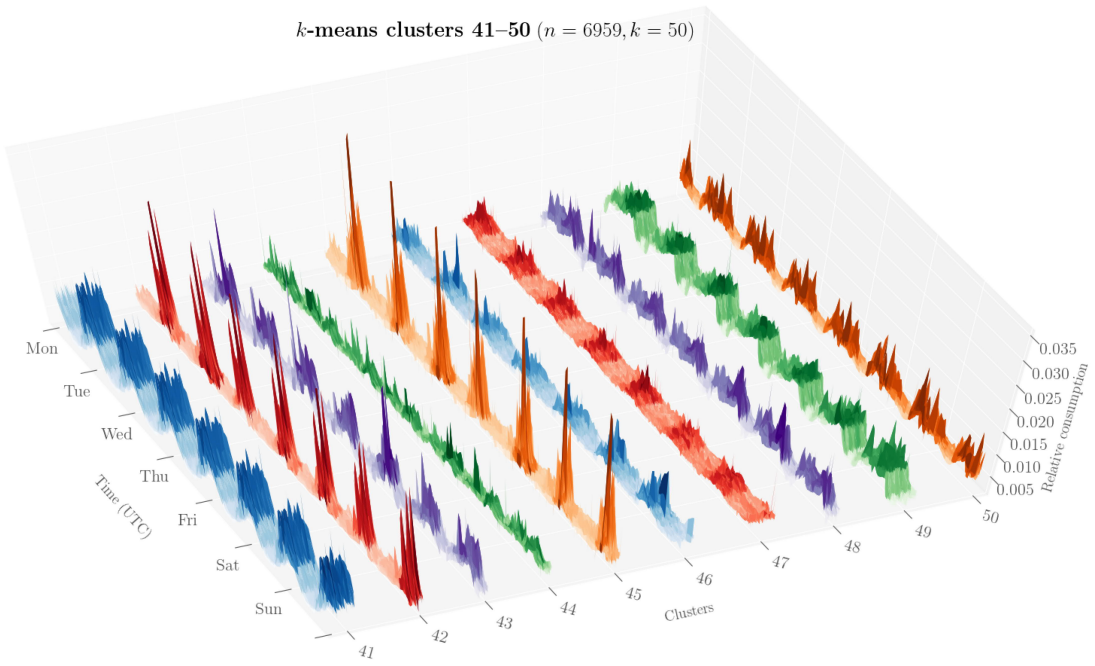

Figure A.4 *The shapes of k-means clusters from* 41 *to* 50.

APPENDIX B. ORCLUS CLUSTERS

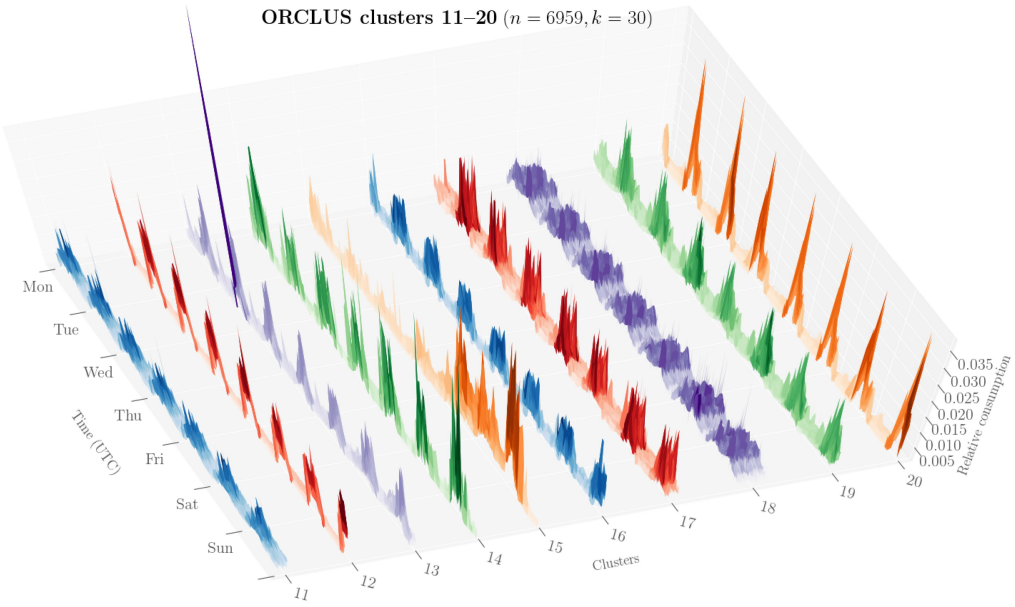

Figure B.1 *The shapes of ORCLUS clusters from* 11 *to* 20.

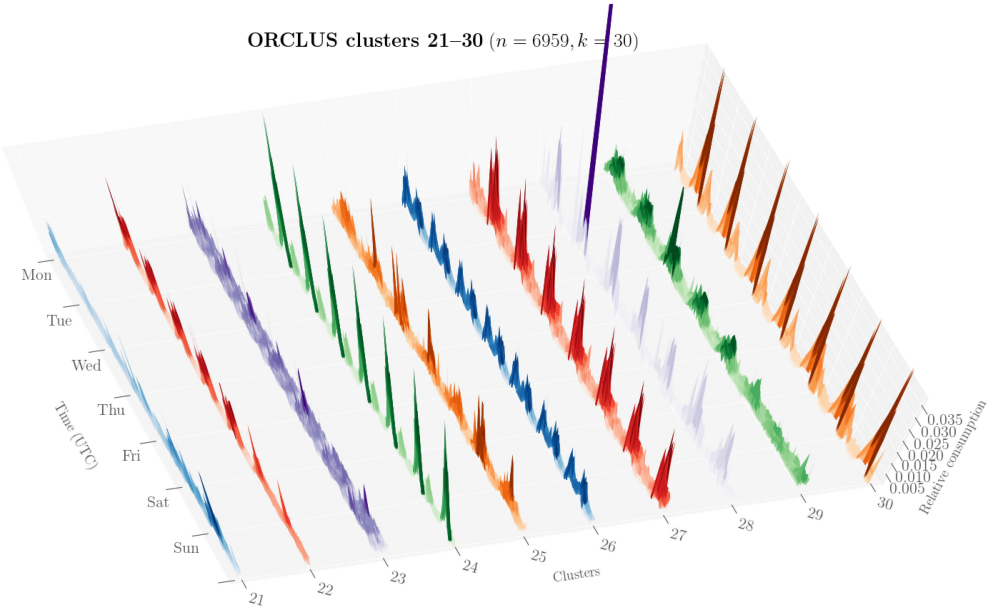

Figure B.2 *The shapes of ORCLUS clusters from* 21 *to* 30.

APPENDIX C. CHUNX AND CRUSHES TREES FOR ENERGY DATA

This appendix contains figures for illustrating constructed CHUNX and CRUSHES trees for real life energy consumption data. Contrary to the format of component labels presented in Figures 3.2 and 3.3, the labels are presented in a format, where the component label is includes the sign of component and the order in the orthogonal basis of the data matrix. Figure C.1 illustrates the CHUNX tree of the energy data and Figure C.2 illustrates the CRUSHES tree. Both figures are presented on separate pages due to their large sizes. Figure C.1 displays 78 clusters from the total number of 178 clusters whereas Figure C.2 displays 99 clusters from the total number of 3875 clusters.

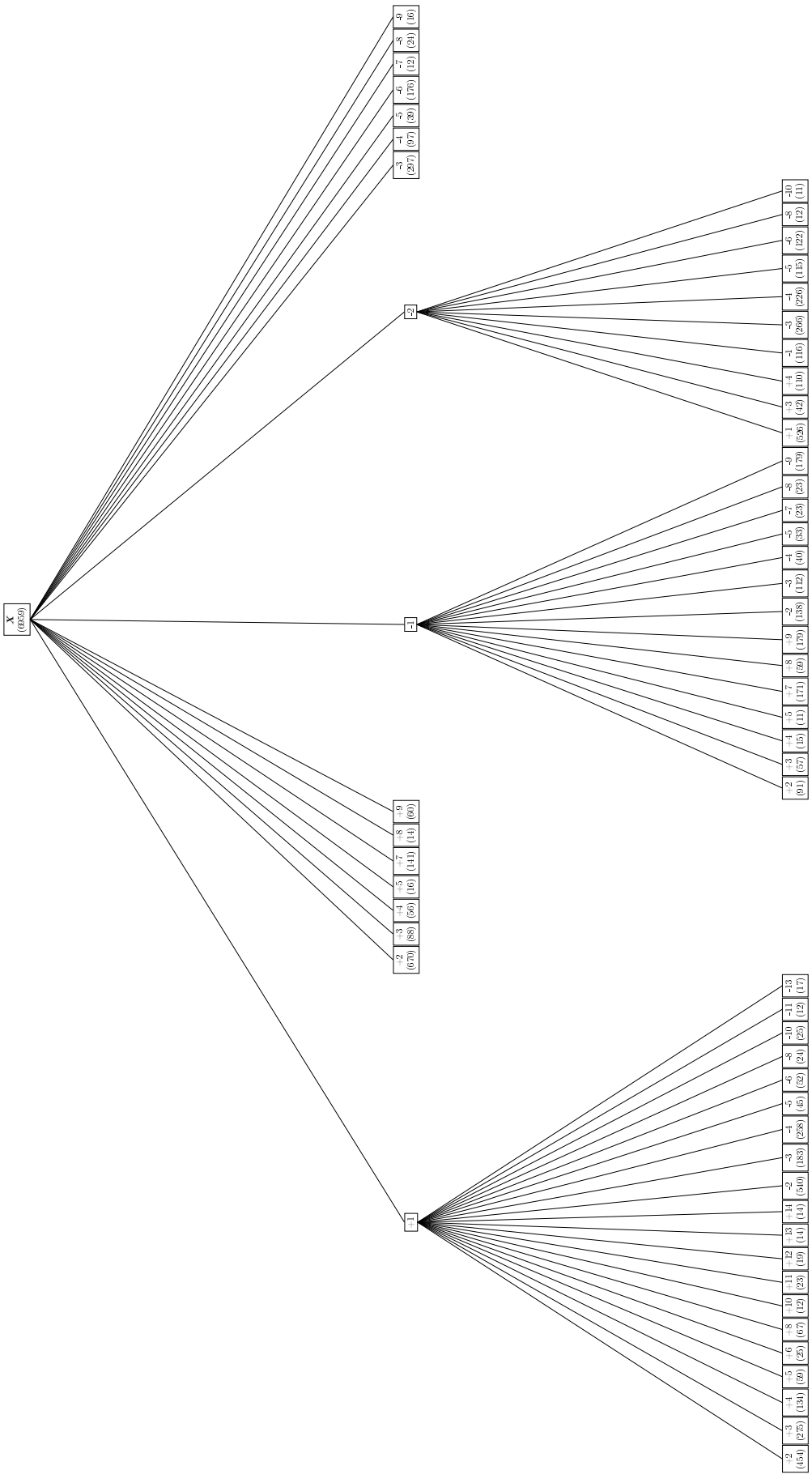

Figure C.1 *CHUNX tree for energy consumption data. The tree is formed with* 0.1 *partition rate for data with* 6959 *samples displaying only clusters with size greater than* 10.

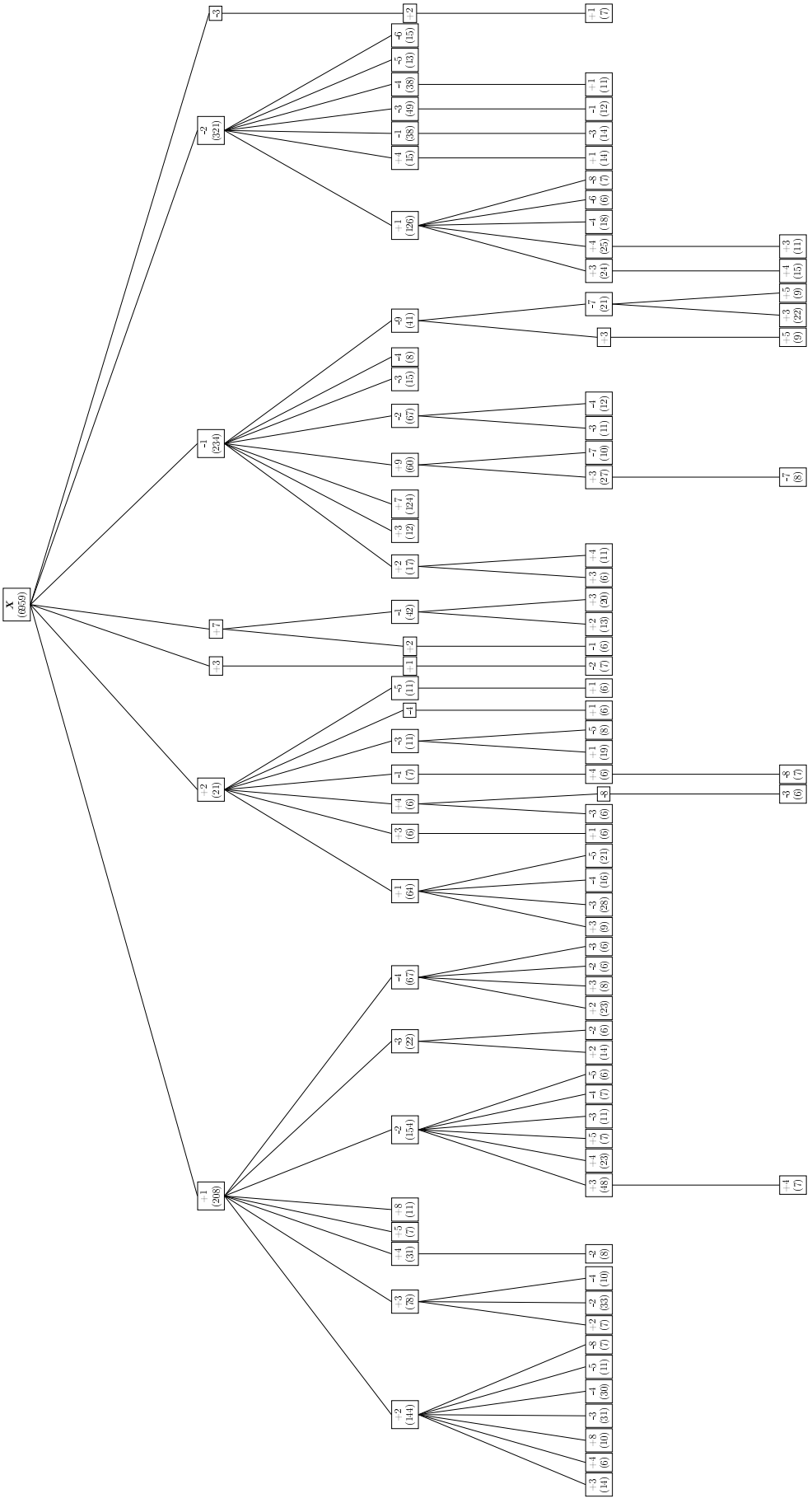

Figure C.2 *CRUSHES tree formed with 0.75 mutual correlation criterion for energy consumption data of 6959 samples displaying only clusters with size greater than 5.*

APPENDIX D. CHUNX CLUSTERS

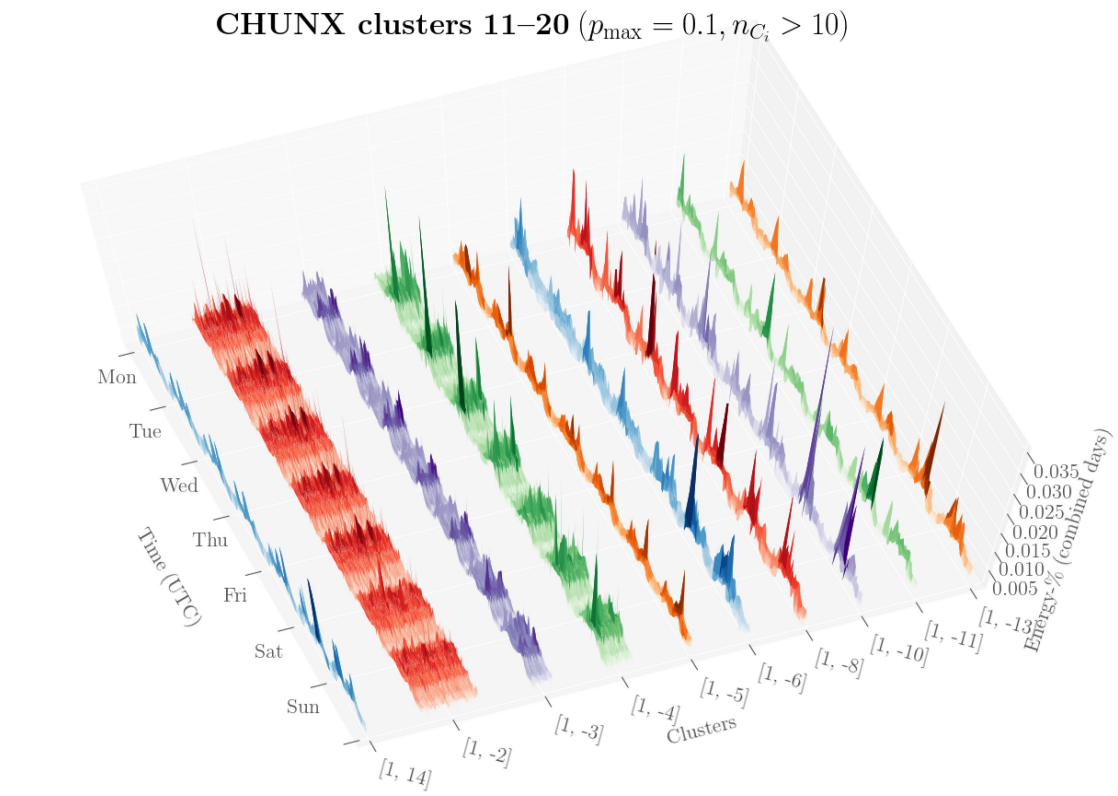

Figure D.1 *The shapes of CHUNX clusters from* 11 *to* 20.

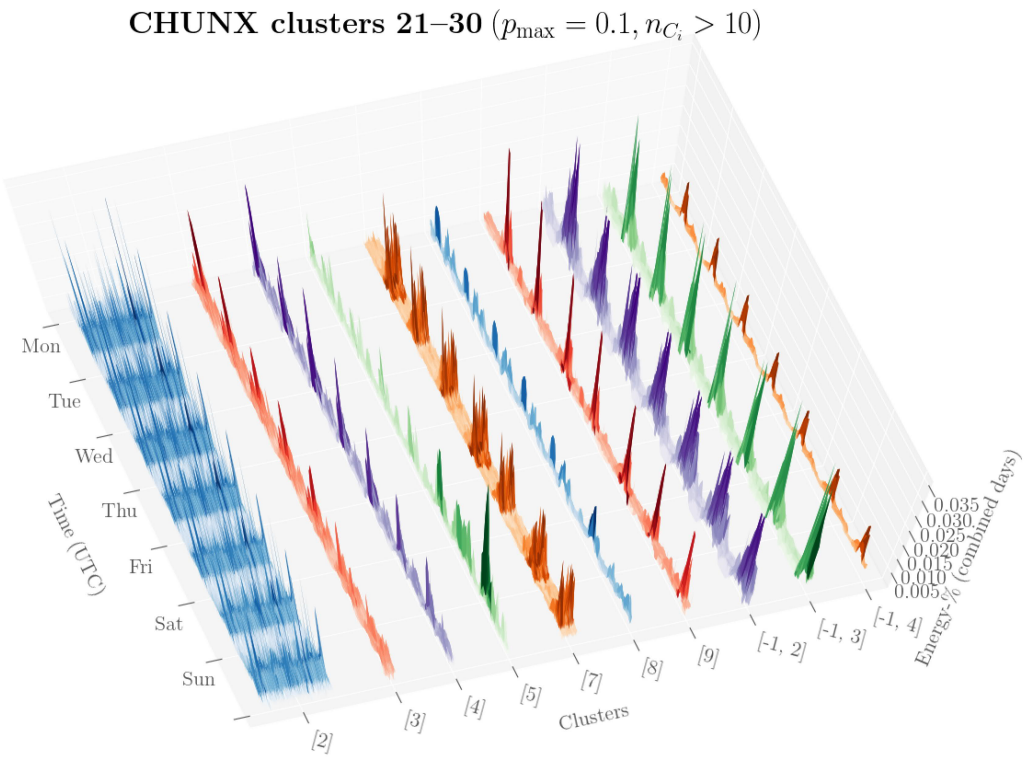

Figure D.2 *The shapes of CHUNX clusters from* 21 *to* 30.

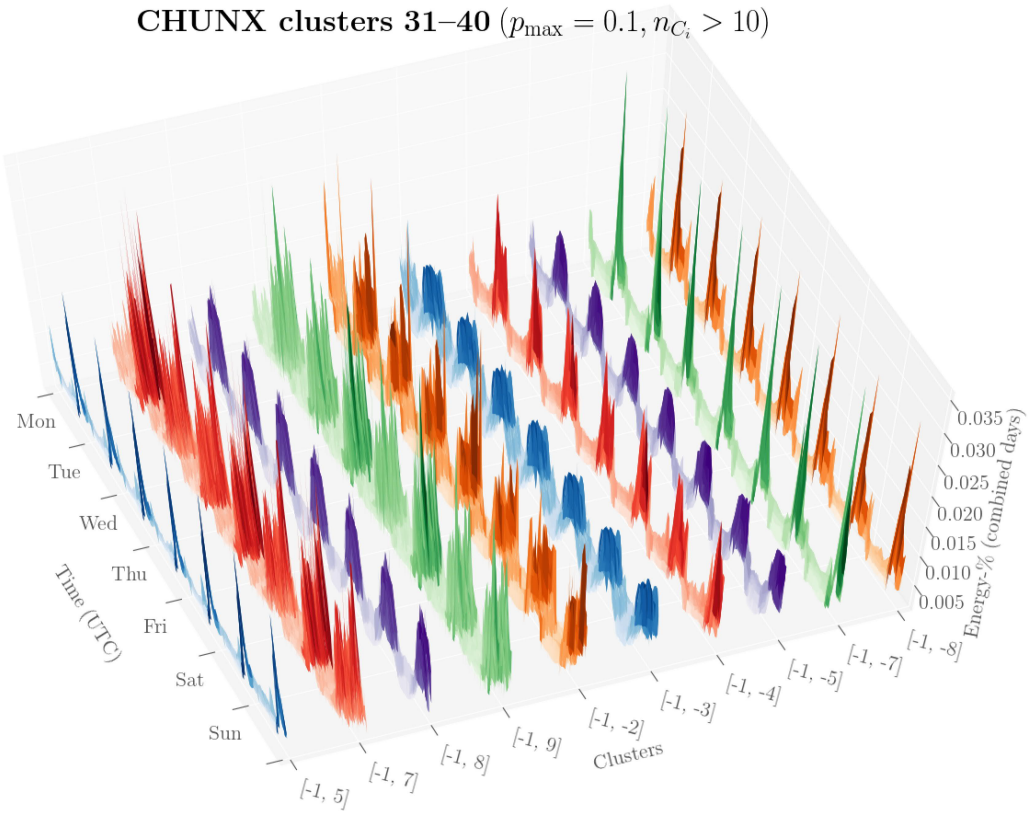

Figure D.3 *The shapes of CHUNX clusters from* 31 *to* 40.

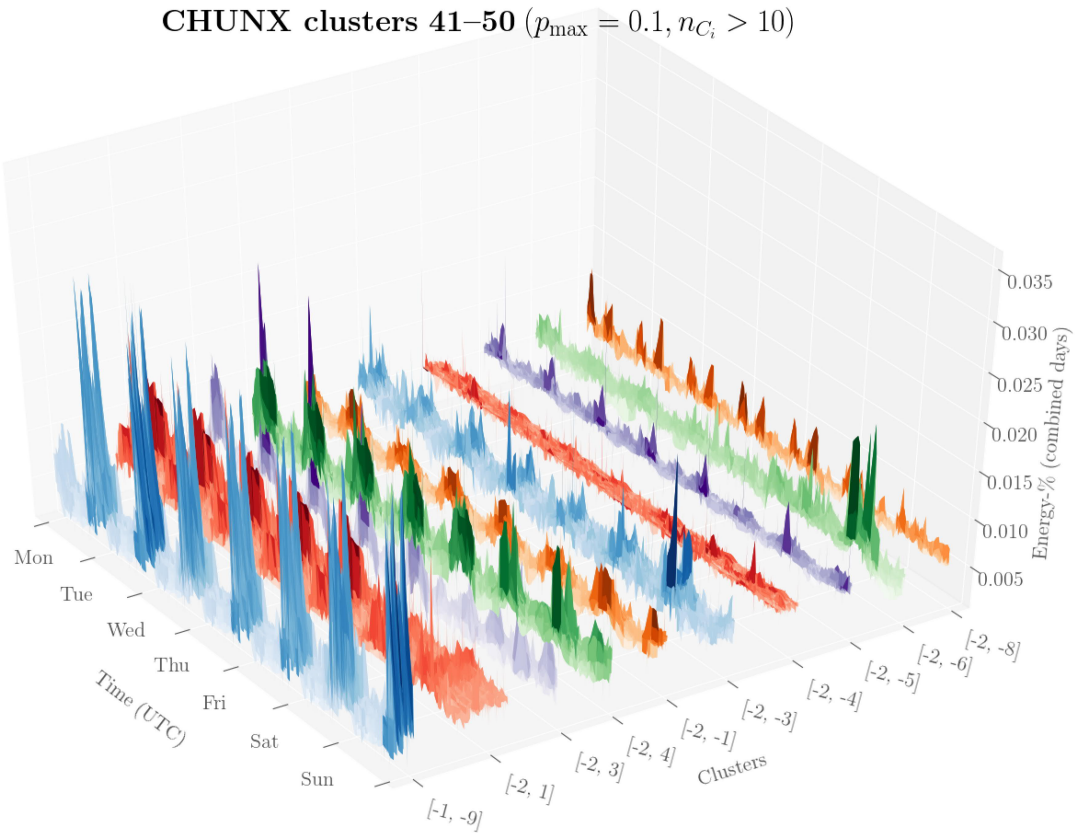

Figure D.4 *The shapes of CHUNX clusters from* 41 *to* 50.

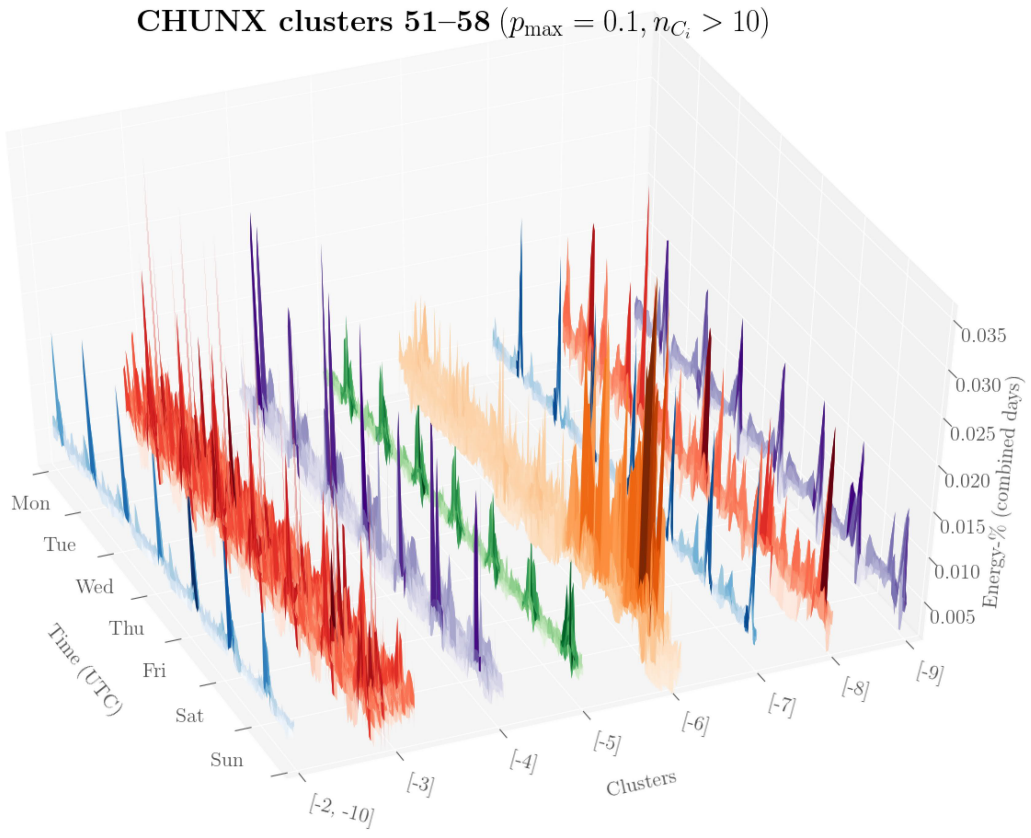

Figure D.5 *The shapes of CHUNX clusters from* 51 *to* 58.

APPENDIX E. CRUSHES CLUSTERS

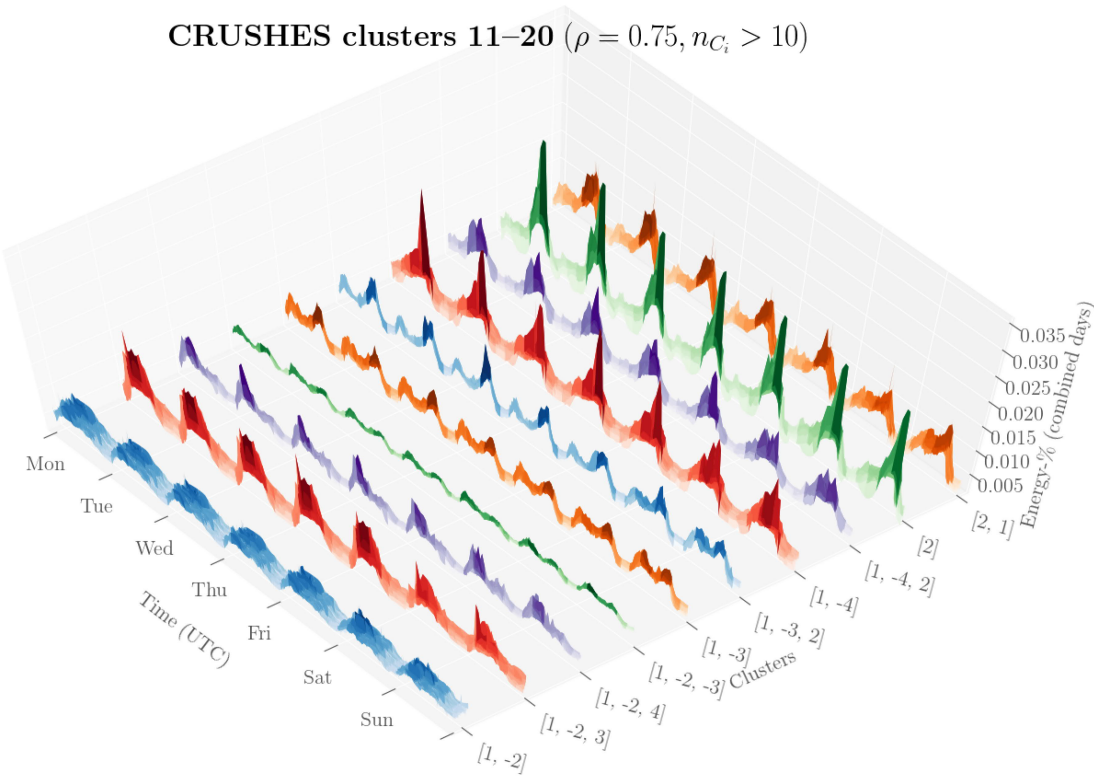

Figure E.1 *The shapes of CRUSHES clusters from* 11 *to* 20.

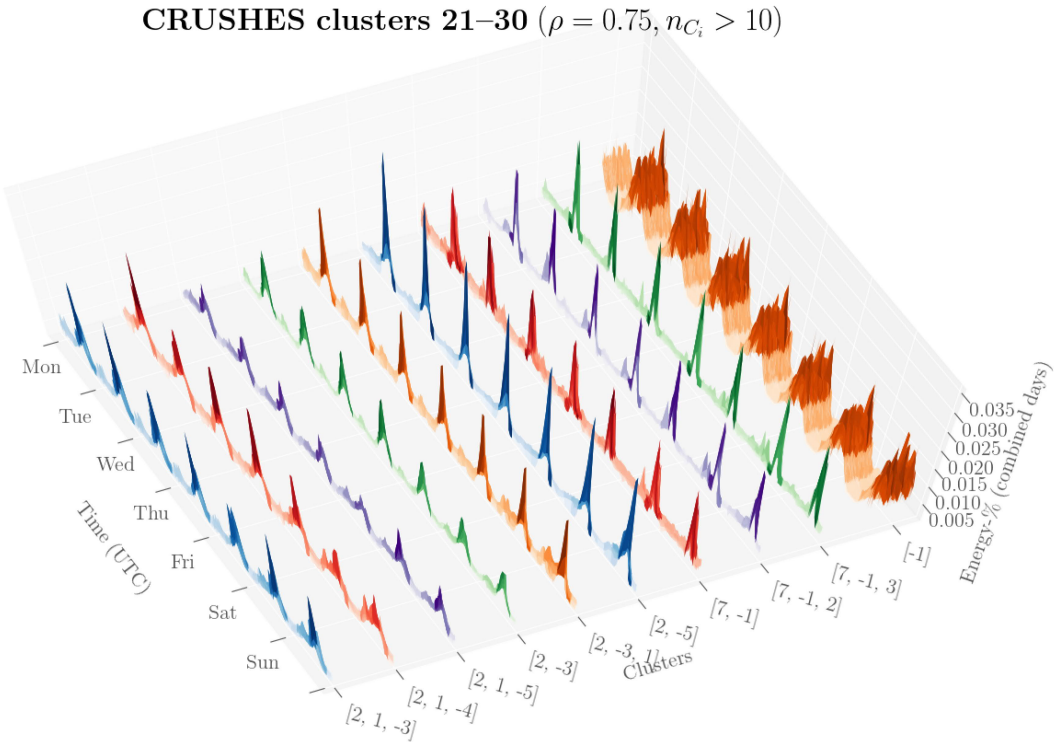

Figure E.2 *The shapes of CRUSHES clusters from* 21 *to* 30.

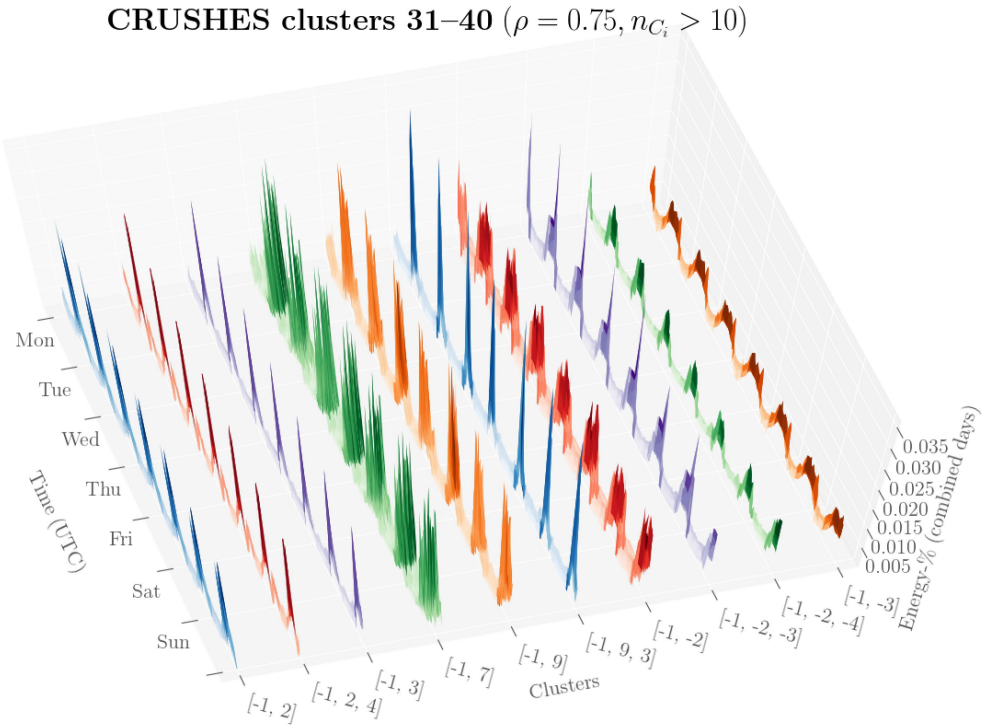

Figure E.3 *The shapes of CRUSHES clusters from* 31 *to* 40.

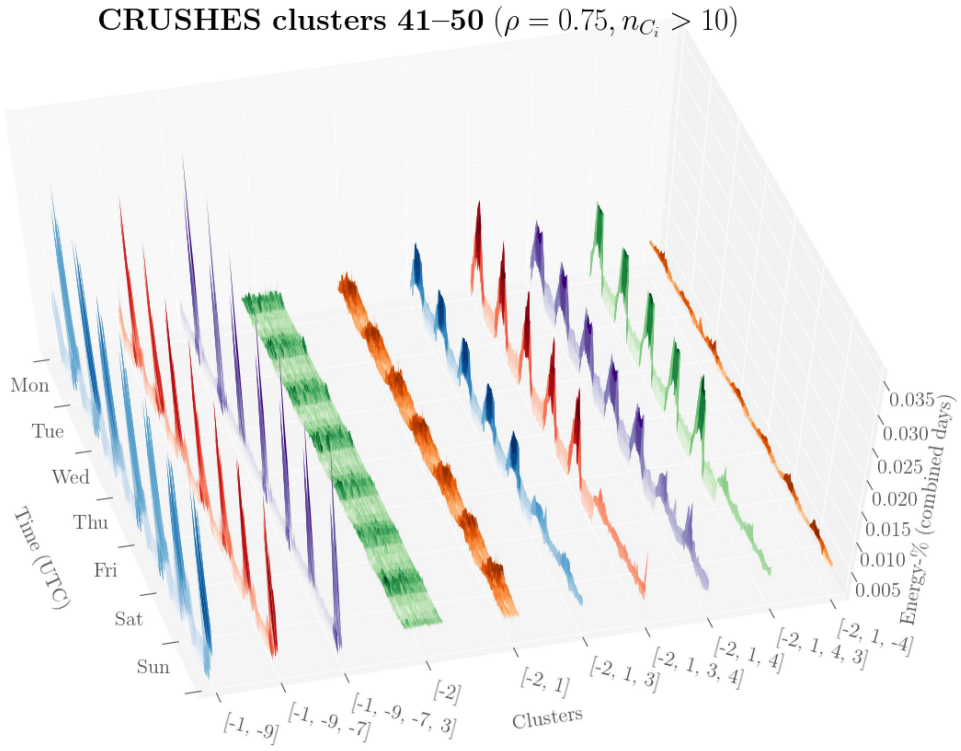

Figure E.4 *The shapes of CRUSHES clusters from* 41 *to* 50.

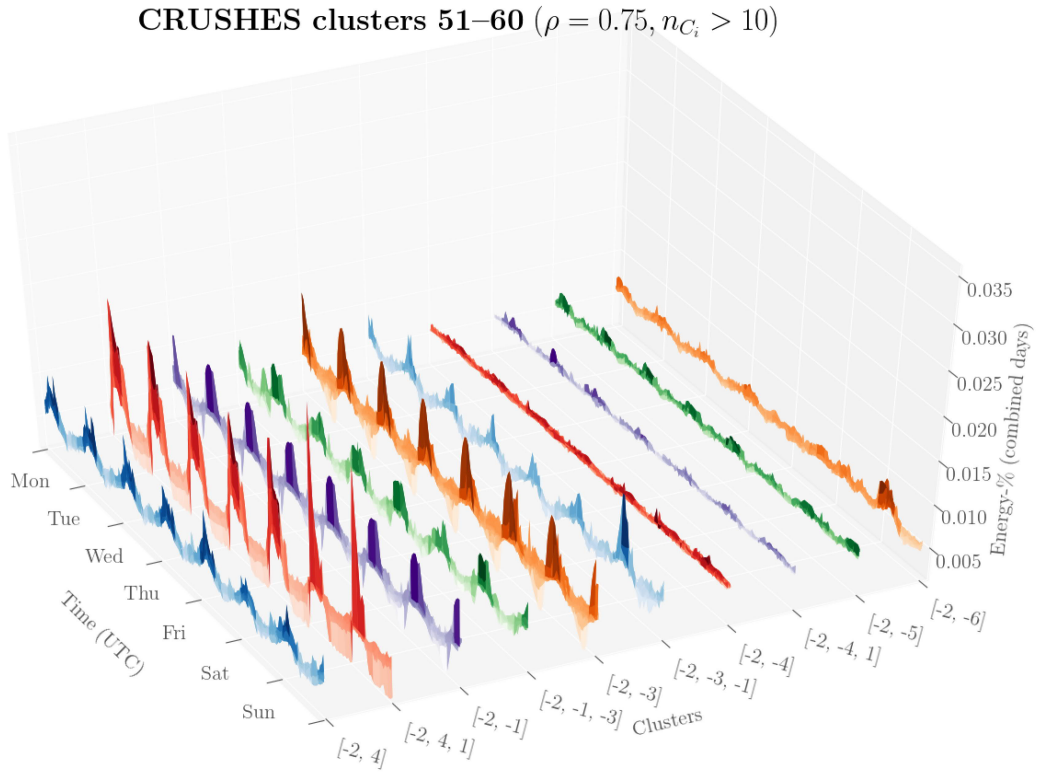

Figure E.5 *The shapes of CRUSHES clusters from* 51 *to* 60.